Ludwig-Maximilians Universität München
Department of Statistics

Master's Thesis

Fake News Detection

Evaluating Unsupervised Representation Learning
for Detecting Stances of Fake News

Author: Maike Guderlei

Supervisor: Prof. Dr. Christian Heumann
Matthias Aßenmacher (M.Sc.)

Advisor: Matthias Wissel (M.Sc.)

Declaration of Authorship

I hereby confirm that I have written the accompanying thesis by myself, without contributions from any sources other than those cited in the text and acknowledgements. This also applies to all graphics, images and tables included in the thesis.

....

(Place and date) (Signature)

Abstract

The aim of this master's thesis is to evaluate the unsupervised representation learning of the five large pretrained NLP models BERT, RoBERTa, DistilBERT, ALBERT and XLNet. In choosing these specific models, the performance of encoder-based models can be evaluated in contrast to an autoregression-based model. The models are examined with respect to two datasets that both consist of instances with a respective headline, associated news article and the stance of the article towards the respective headline. The headline states a short claim and the article bodies either *Agree* or *Disagree* with the stated claim, *Discuss* it or are simply *Unrelated*. The datasets are introduced in the scope of the Fake News Challenge Stage 1 that took place in 2017. Specifically, the thesis aims at understanding how much hyperparameter tuning is necessary when finetuning the five models, how well transfer learning works in the specific context of stance detection of Fake News and how sensitive the models are to changing the hyperparameters batch size, learning rate, learning rate schedule, sequence length as well as the freezing technique. Furthermore, the finetuning for large pretrained NLP models is evaluated by opposing the encoding- to the autoregression-based approach.

The results indicate that the much more expensive autoregression approach of XLNet is outperformed by BERT-based models, notably RoBERTa. The encoding approach thus yields better results. The hyperparameter learning rate is most important, while the learning rate schedule is relatively robust to changes. Experimenting with different freezing techniques indicates that all models learn powerful language representations that pose a good starting point for finetuning. Using the larger FNC-1 ARC dataset that is more evenly distributed boosts prediction performance especially for the sparse category of *Disagree* instances for most models.

List of Figures

List of Tables

List of Acronyms

AI	Artificial Intelligence
NLP	Natural Language Processing
NLI	Natural Language Inference
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
seq2seq	Sequence-to-Sequence
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
FF	Feed-Forward Layer
TF-IDF	Term Frequency-Inverse Document Frequency
SVD	Singular Value Decomposition
MLM	Masked Language Model
PLM	Permutation Language Model
NSP	Next Sentence Prediction
SOP	Sentence Order Prediction
[MASK]	masking token
[UNK]	unknown token
[SEP]	separation token
[CLS]	classification token
FNC-1	Fake News Challenge Stage 1
ARC	Argument Reasoning Comprehension
GLUE	General Language Understanding Evaluation
SOTA	state-of-the-art

Content

1 Introduction

With the rise of social media, exchange of opinions and news happens faster than ever. News circulation is therefore less and less bound to traditional print journalism that usually requires extensive research, fact checking and accurate coverage in order to be a reliable news resource. It is relatively easy to share opinions that are either not supported by researched facts or simply wrong. In the worst case a large amount of people can be targeted by propaganda in order to shift societal discussions in favor of a wanted agenda. Human resources are limited to identify such Fake News, since they cover a wide range of topics and linguistic writing styles (Shu, Sliva, et al. 2017, p.2). Automated Fake News detection has therefore proven to be an important challenge for NLP researchers in recent years.

To this day, a variety of approaches to deal with Fake News detection exist (Khan et al. 2019). One approach deals with the Fake News detection task from a stance perspective. The idea is to determine the stance of a news article to a given headline (Hanselowski et al. 2018, p.1). In 2017, the Fake News Challenge Stage 1 was introduced by a team from academia and the industry tackling Fake News detection as a stance detection task (Pomerleau and Rao 2017). As the Fake News detection task is a difficult one, the initiators decided to break it down into a pre-step of identifying Fake News by understanding what other news say about the same topic. The stance of an article is then one of the four categories *Agree*, *Disagree*, *Discuss* and *Unrelated*.

This thesis evaluates the five models BERT (Devlin et al. 2018), RoBERTa (Yinhan Liu et al. 2019), DistilBERT (Sanh et al. 2019), ALBERT (Lan et al. 2019) and XLNet (Z. Yang et al. 2019) that have been developed and enhanced recently. All five models have learned powerful represenations of language on large corpora and can be finetuned on a desired task. The models are evaluated in the context of the above discussed Fake News Challenge. The main focus is to evaluate the necessity of tuning hyperparameters as well as the general performance. For evaluating the general performance, it is of special interest to examine the differences between the encoding-based models (BERT and BERT-based architectures) in contrast to the autoregression-based model XLNet.

The thesis is outlined by first discussing the term *Fake News* and giving an overview of exisiting datasets and approaches to Fake News detection in chapter 2. Fake News is defined as a text piece that is verifiably wrong and shared with malicious intentions. Furthermore, Fake News is introduced in distinction to related terms such as clickbait, rumors, conspiracy theory and hoaxes. Chapter 3

outlines the main theoretical assumptions and the neural network's architectures of BERT, RoBERTa, DistilBERT, ALBERT and XLNet. A focus is put on understanding how RoBERTa, DistilBERT and ALBERT have further boosted and enhanced the already extraordinary performance of BERT as well as the distinction between the encoding approach of BERT versus the autoregressive approach of XL-Net. It will become clear how differently the models are pretrained with respect to used data, tokenization algorithms and further hyperparameter choices. After introducing the Fake News Challenge in chapter 4 as well as the published baseline and winning models, the two datasets that are used for evaluating the five models are made known in chapters 5.1 and 5.2. The main pre-processing steps are presented and explained in chapter 5.4. After having understood the background of the two datasets as well as the data pre-processing, the choice of the evaluation metric is given in chapter 5.5. The proposed evaluation metric of the FNC-1 organizers can not adequately handle the uneven distribution of the four categories and is thus replaced by a macro-averaged F_1 metric. The final main chapter 6 starts by giving details about the implementation and then reflects on benchmarking in NLP. The generated insights are used to choose the first set of hyperparameters for the setup of initial experiments. In these first experiments, the general performance of the five models is evaluated with respect to three different freezing techniques and overfitting tendencies. Building up on this first evaluation step, an extensive grid search over the learning rate, learning rate schedule, batch size and sequence length is performed resulting in 48 combinations to be evaluated per model and dataset. The thesis ends with summing the key aspects up, drawing conclusions from the results to answer to posed questions and giving an outlook on the further disguise of Fake News detection research.

One of the biggest problems with Fake News is not necessarily that it gets written but rather that it gets spread.

Vasandani

2 Fake News Detection

2.1 Definition of Fake News

In 2017 Facebook published a whitepaper discussing potential threats for online communication and the responsibility that comes along with being one of the largest social media platforms of these days (Weedon, Nuland, and Stamos 2017). Weedon, Nuland, and Stamos furthermore discuss the emerging problem of using the blurred term of Fake News, stating that "the overuse and misuse of the term "fake news" can be problematic because, without common definitions, we cannot understand or fully address these issues." (Weedon, Nuland, and Stamos 2017, p.4). The term can refer to anything ranging from factually wrong news articles, to hoaxes, april fools, rumors, clickbait or stated opinions shared online that contain wrong information.

In this thesis, the term Fake News is defined as a text piece that is verifiably wrong and spread with a malicious intention. There are three key aspects to this definition. Firstly, by specifically referring to textual Fake News, other media sources such as video, images or audio are excluded. Tackling the detection of so called "deep fakes" requires different AI solutions than working with text. As a second important aspect, the definition implies that Fake News can be fact-checked. It is thus possible to verify the stated claim(s) as either true or false. In incorporating this into the definition, rumors are excluded since it is often impossible to verify them. Conspiracy theories fall into the category of rumors since they can be seen as a long-term rumor that states claims which are hard to distinctly refute. Lastly, the definition targets the intention of Fake News. Since this intention has to be malicious all sorts of entertainment related false news such as hoaxes and april fools are excluded. Furthermore the intention is meant to be malicious in the sense of wanting to influence societal discussion often in favor of a certain propaganda. This also excludes unintentionally wrong published text pieces with for example transposed digits. The definition is closely linked to the narrow definition that Shu, Sliva,

et al. undertake.

Another important distinction is the one between Fake News and clickbait. Clickbait can be described as attention-grabbing headlines that create a curiosity gap. When clicking on such a headline, the reader is typically redirected to another website with poorly reasoned article bodies that sometimes don't have much to do with the suggested topic of the headline. The goal of clickbait however is to generate revenue by increasing traffic on the website which usually contains advertisement. The intent is mainly financial and not a malicious propaganda spread to promote a certain political agenda. Clickbait detection focuses on modeling the relationship between a headline and a news article. In chapter 4 this idea will be further exploited and adapted to the context of Fake News detection.
A typical example of Fake News is the false news that was systematically disseminated by Russian trolls about Hillary Clinton during the election campaign for the U.S. presidency in 2016 in order to shift people's voting tendency toward Donald Trump (Mueller 2019). In this case it is evident how harmful Fake News can be. But there is an additional problem to the topic of Fake News. Some Fake News are simply spread for the sake of triggering people's distrust, to sow confusion and to hinder people from being able to clearly distinguish between what is true and what is false.
In order to fully understand this problem, Shu, Sliva, et al. propose to consider the ecosystem of information sharing within the prospect theory paradigm (Kahneman and Tversky 1979). In prospect theory decision making is modelled as process in which people make choices based on relative gains and losses compared to the current situation. These relative gains and losses should be understood in the context of general psychological and sociological phenomenons that also apply during the process of news consumption. For example, people gain something by adhering to "socially safe" options. In the context of news consumption this means that people tend to follow norms that have been established in their community *even if the news that are shared are Fake News* (Shu, Sliva, et al. 2017, p.3). In addition, it is a general challenge for humans to filter out Fake News from real news, since Fake News cover almost any topic and can imitate persuasive and concise writing styles. Furthermore, there are factors that make people even more susceptible to Fake News. One of these factors is the tendency of people to believe that their perceptions of reality are the only accurate views. Other people who disagree are then seen as uninformed, irrational or biased. One of the most common biases when it comes to information processing is the so called confirmation bias which leads people to selectively put attention onto information pieces that confirm their already existing beliefs. The problem of confirmation bias in the social media context can be furthermore described as the echo chamber effect. The echo chamber effect describes how existing beliefs are reinforced by communication and repetition inside a closed system. In the context of Fake News this closed system can be seen as such a simple thing as the social media homepage of a user that is adjusted to the user's assumed preferences by an underlying algorithm. A user is shown more articles that confirm his or her beliefs when spending time on social media, which amplifies the already existing confirmation bias and makes the spread of Fake News even easier. These phenomenons are so strong that there are even indications that correcting Fake News by showing the actual true information might not only be

unhelpful in mitigating misperceptions but could actually increase them (Nyhan and Reifler 2010).

To put these different aspects together, one can think of a news generation and consumption cycle with two parts. On one hand, there is the publisher of news who has a short- and a long-term interest. The short-term interest is to maximize profit which usually translates to increasing the number of reached consumers. The long-term interest is to achieve and keep a good reputation with respect to news authenticity. On the other hand, there is the consumer side. The consumer aims at fulfilling two needs. The first one is to obtain true and unbiased information while the other one is to receive news that satisfy and align with prior beliefs and social needs. These latter needs can be explained by the confirmation bias and the social identity theory which states that social acceptance and affirmation is essential to a person's identity and self-esteem. Fake News can thrive in an environment where publishers are driven by a short-term interest while consumers' needs are dominated by psychological needs rather than the need for unbiased and true information. This is important to understand because simply detecting, classifying or reporting Fake News does not consider the social dynamics that facilitate the dissemination in the first place and the fact that Fake News does not only serve a malicious publisher but also fulfills basic human needs of consumers. Creating AI tools that can deal with Fake News is therefore only the surface of a much larger discussion on how to deal with them.

2.2 Related work

After having elaborated on the general background of Fake News and the important aspects related to the use of this term, a brief overview of existing approaches to Fake News detection is given.
Fake News detection can be interpreted as a classification task where a text document is classified as either *Fake News* or *No Fake News* or as a multiclass problem often with ordinal labels. The main challenge in finding a suiting classification algorithm lies in the necessary data basis. Currently there are variety of different datasets available for Fake News detection. The most important ones are the dataset released by Vlachos and Riedel in 2014, the LIAR dataset (W. Y. Wang 2017), the dataset used for the Fake News Challenge Stage 1 which will be introduced in more detail in chapter 4, the FakeNewsNet repository which contains two datasets as well as the FEVER dataset (Thorne et al. 2018).
The first publically released dataset tackling Fake News detection was published in 2014 by Vlachos and Riedel. The authors took two fact-checking websites as data basis and assigned a consistent labeling scheme. The first one is the British Fact Checking website of Channel 4 (Channel 4 2020) and the second one the Truth-O-Meter of the website `politifact.org` (Times 2020). PolitiFact is a running project where journalists and domain experts review political news and evaluate them with respect to their truthfulness (Shu, Mahudeswaran, et al. 2018, p.3). The data thus contains a variety of prevalent topics of public life in the U.K. and U.S.. Fact-checking is not understood as a binary task, since the extent to which statements are false can vary. The statements are thus labeled as

True, *Mostly true*, *Half true*, *Mostly False* and *False*. Statements that were classified relying on data that was not available online were excluded. Therefore out of 221 considered statements only 106 statements were chosen for the dataset. Since the dataset is so small its capability is limited in using modern machine learning algorithms.
In 2017, W. Y. Wang introduced a new benchmark dataset called LIAR. It contains 12,836 manually labeled statements and spans a decade of time. LIAR also relies on PolitiFact as main data source and uses statements of politicians. The instances are labeled for truthfulness, subject, speaker, state, party, prior history (counts of already inaccurate statements of the respective person) and context, such as the current job of the person. The authors use a labeling scheme of six labels, namely *Pants-fire*, *False*, *Barely-true*, *Half-true*, *Mostly true* and *True*, which are relatively balanced.
In the same year, Pomerleau and Rao introduce the Fake News Challenge Stage 1 which introduces a stance-based dataset of headlines and article bodies. The FNC-1 dataset as well as the extended FNC-1 ARC dataset are subject to a more detailed description in chapters 5.1 and 5.2. As a short notice, the FNC-1 relies on Emergent (Ferreira and Vlachos 2016), a dataset that also uses PolitiFact as a main source, while the extended FNC-1 ARC dataset additionally considers user posts of the online debate section of the New York Times.
A year later, Thorne et al. introduced FEVER, a large-scale dataset for Fact Extraction and VERification. FEVER covers 185,445 claims that are annotated with the three labels *Supported*, *Refuted* and *Not Enough Info*. In contrast to the other datasets, FEVER does not rely on PolitiFact but uses Wikipedia as data source. The data was generated by first extracting information from Wikipedia from which claims are engendered. The second step is to then annotate the claim with the corresponding class.

As last interesting project, the FakeNewsNet[1] is worth mentioning which is a Fake News data repository with two datasets. The datasets do not only contain annotated statements but also incorporate spatiotemporal information and information on social context. In addition, Shu, Mahudeswaran, et al. introduce a pipeline of continually updating data for current Fake News. The pipeline can be seen in figure 2.1. The news content relies again on PolitiFact and also on GossipCop. By incorporating GossipCop (Shuster 2020), a website that fact-checks claims concerning celebrities, more mundane false news is taken into consideration that doesn't strictly fulfill the aforementioned criteria of Fake News[2].

In addition to these fairly popular Fake News datasets, an array of smaller and less known datasets is available. These are BuzzFeedNews (BuzzFeed 2020), B.S. Detector (Sieradski 2020), CREDBANK (Mitra and Gilbert 2015), BuzzFace (Santia and Williams 2018) and FacebookHoax (Tacchini et al. 2017), to only name a few.
BuzzFeedNews contains news that were published on Facebook from 9 different news agencies. The

[1]The project is available online via Shu 2020.
[2]See chapter 2.1 for the used definition of Fake News for this thesis

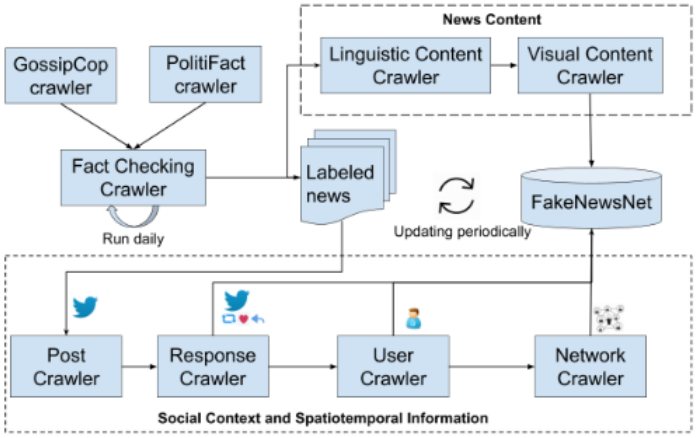

Fig. 2.1: The pipeline of the data integration process for the FakeNewsNet repository. The news content is based on crawling the fact-checking websites GossipCop and PolitiFact. In addition, social context and spatiotemporal information is integrated by crawling Twitter for the discussed headlines. For the user's engagement, location and timestamps are crawled.
Source: Shu, Mahudeswaran, et al. 2018, p.3.

timespan coves a week close to the U.S. elections for the presidency in 2016. The dataset consists of 1,627 articles that were manually fact-checked by BuzzFeed journalists. BuzzFace is an extension of this dataset and uses 2,263 news articles and 1.6 million comments discussing these news. B.S. Detector is based on a browser extension with the same name. Using this browser extension, a webpage is crawled for all used links and checks them against a manual list of domains. This yields machine annotated labels rather than annotations made by humans. CREDBANK covers 60 million tweets over the course of 96 days starting from October 2015. The tweets were associated with more than 1,000 news events and each event was manually annotated with respect to credibility. FacebookHoax contains 15,500 posts from 32 Facebook pages. The pages are divided into non-hoax (id est science-related news) and hoax pages (id est conspiracy theory news).
Not all datasets match the given definition of Fake News. Furthermore the mentioned datasets are not to be understood as an exhaustive list over every ever published dataset affiliated with the broad topic of Fake News detection but rather as a broad overview. One interesting aspect is that all major benchmark datasets have annotated the instances with a finer-grained scale than simply *Fake News* versus *No Fake News*. It seems plausible to use nuances of "fakeness" given the very different possible publishers and media platforms of Fake News. Furthermore, many datasets rely on PolitiFact.

If the datasets concerning Fake News detection are already versatile, the approaches on how to actually model a detection algorithm are even more so. One major distinction that can be drawn is the consideration of auxiliary information. Most contributions take a purely feature-oriented approach, while others try to incorporate information revolving around spreading Fake News and

the respective dissemination process. The feature-based approach focuses on extracting relevant linguistic-based peculiarities associated with Fake News. These features can be lexical or semantical. Typical examples are n-grams of characters or words or a measure of the readability and syntax of an article body (Pérez-Rosas et al. 2018, p.5). Other authors have included the average word length, counts of exclamation marks or the sentiment of an article (Khan et al. 2019, p.6). The key takeaway is that feature extraction is based on the text piece itself. After finding appropriate features, traditional machine learning algorithms as well as (deep) neural network architectures are proposed to classify the text instance given the extracted features. Khan et al. specifically compare traditional algorithms such as logistic regression[3], support-vector machine[4], decision trees[5], naïve bayes[6] and k-nearest neighbor[7] to a wide range of neural architectures based on CNNs[8] or LSTMs[9]. The naïve bayes classifier works surprisingly well, while the performances of the neural networks largely depend on the underlying dataset. Pérez-Rosas et al. also implement a support-vector machine algorithm. Ni et al. take a different approach by trying to detect good generalizable features with propensity score matching[10] that are then evaluated by logistic regression, support-vector machine and random forests[11]. Another direction of the feature-based approach is taken by some researchers who implement neural network architectures. When relying on deep learning, the model can also just learn a good representation of the text input by a stack of hidden layers that is then used for the last classification layer. This concept of representation learning along with the necessary terminology is introduced in more detail in the following chapter 3. Y. Yang et al. take this approach by simultaneously training a CNN on text *and* image data to classfiy fake entities or by Dong et al. who implement a two-step approach of using supervised and unsupervised learning with a CNN as well.

Although, feature-based approaches are fairly popular within the Fake News detection research, Shu, Sliva, et al. argue that they are not sufficient. In light of the already discussed ecosystem of information sharing this remark is important. However, approaches that use auxiliary information are less common. Usually they try to model the dissemination process of Fake News by incorporating spatiotemporal information about users who like, share or publish (potential) Fake News. Two examples of this network-based approach[12] are given by Ren and Zhang and Ruchansky, Seo, and Yan Liu. Ren and Zhang propose a hierarchical graph attention network that learns information from different types of related nodes through node- and schema-level attention, while Ruchansky, Seo, and Yan Liu capture temporal patterns on user activities on a given article with an RNN.

The Fake News Challenge Stage 1, introduced in more detail in chapter 4, uses a feature-based

[3]For more details, go to Hastie, Tibshirani, and Friedman 2009, p.119.
[4]For more details, go to Hastie, Tibshirani, and Friedman 2009, p.417.
[5]For more details, go to Hastie, Tibshirani, and Friedman 2009, p.305.
[6]For more details, go to Hastie, Tibshirani, and Friedman 2009, p.210.
[7]For more details, go to Hastie, Tibshirani, and Friedman 2009, p.463.
[8]For more details, go to Goodfellow, Bengio, and Courville 2016, p.321.
[9]For more details, go to Goodfellow, Bengio, and Courville 2016, p.397.
[10]For more details, refer to Rosenbaum and Rubin 1983.
[11]For more details, go to Hastie, Tibshirani, and Friedman 2009, p.587.
[12]In this case, the term network does not refer to the general architecture of deep learning models (neural networks) but means the network revolving around the spreading of (potential) Fake News.

approach by proposing a baseline model that extracts various features from the headlines and article bodies. The approach of specifically modeling the relationship between a headline and a respective article body was also exploited by Yoon et al. but does so in the context of clickbait detection while the FNC-1 takes this approach as a pre-step of Fake News detection.
BERT, RoBERTa, DistilBERT, ALBERT and XLNet make use of the mentioned second direction within the feature-based approach by learning powerful representations of textual input, as explained in the following chapter 3.

3

Unsupervised Representation Learning

Representation learning can be seen as one of the crucial success factors of large pretrained models such as BERT or XLNet that yield outstanding performances on a variety of NLP tasks. In fact, representation learning is discussed as being the cause for the newly developed interest in using (deep) neural network architectures that has emerged since 2006.
In this chapter the terminology of (unsupervised) representation learning, transfer learning, pretraining and finetuning are defined. In addition, the model architectures to be evaluated, namely BERT, RoBERTa, DistilBERT, ALBERT and XLNet as well as the necessary background of the Transformer architecture are introduced. Since language models form the basis of the transformation from simply learning word embeddings to now having powerful network architectures that excel outstandingly on a variety of NLP tasks, the definition of them will be given as well.

In general, any neural network learns a representation of the given input that allows for the best possible performance on a given task. The goal of a deep feed-forward neural network is to approximate some function f^*. If the model is trained on a classification task, $y = f^*(\mathbf{x})$ maps an input $\mathbf{x}$ to a category y. In the case of NLP, the input usually consists of a sequence of words[1], id est $\mathbf{x} = \left(x_1, ..., x_T\right)$. A typical classification example is to categorize a given text input $\mathbf{x}$ according to its sentiment. Or the text might be an email and the classification task is then to find out, whether the email can be classified as spam or no spam. In both cases, a feed-forward model defines the mapping $\mathbf{y} = f(\mathbf{x}, \theta)$ with $\mathbf{x}$ being the text input and θ the learned weights of the network. The weights are learned such that a good function approximation of the unknown function f^* is achieved (Goodfellow, Bengio, and Courville 2016, p.164). The key of deep learning algorithm is that the learned function $f(\mathbf{x})$ usually consists of a composition of functions which means that $\mathbf{x}$ gets processed by a number of functions. One can think of three functions f^1, f^2 and f^3 and a three-fold composition $f(\mathbf{x}) = f^3(f^2(f^1(\mathbf{x})))$ with f^1, f^2 and f^3 being referred to as the first, second and third layer respectively. Usually it is the last layer that provides some task-specific action like calculating the probability distribution over multiple classes via softmax[2], while all other layers in a deep network simply learn a good representation for this classifier. This implies that the choice of representation is not really of interest, since it's main goal is to make the subsequent learning task easier (Goodfellow, Bengio, and Courville 2016, p.524ff.). Staying within

[1] The specifics of input in NLP will be discussed later on. For now, input shall be defined as word sequence for demonstration purposes.
[2] Go to appendix A for more details.

the example of a classification task in the form of sentiment analysis, f^1 and f^2 would learn a good representation of the text input, while f^3 would be the final classification layer. Since the outputs of f^1 and f^2 are not really of interest, they are also called hidden layers with respective hidden states. The hidden states are usually denoted by h with h_t being the hidden state of hidden layer t if not otherwise stated[3]. The representation is therefore learned as a side-effect. In addition, a big advantage of representation learning is the possibility to train a model with unlabeled data (unsupervised learning) which is crucial in a reality where labeled data is sparse and expensive but unlabeled text is easily available. Transfer learning is the process of exploiting the learnings of a specific setting to improve generalization in a different setting. A model could be initially trained for neural machine translation but learn inherent language structures that are useful for other NLP tasks such as question and answering. The learning algorithm used in transfer learning can thus be used for solving two or more different tasks. The underlying assumption in transfer learning is that the data distributions for the different tasks can be explained by some shared factors (Goodfellow, Bengio, and Courville 2016, p.534). In this setting, all considered models fall under this category of transfer learning. Each model is trained on large corpora (unlabeled data), with a neural network architecture learning internal representations of these text data. The model's parameters are pretrained on this large corpora and can be finetuned for a variety of different NLP tasks. Pretraining is therefore the first step of attaining a large model structure with trained weights in all layers. As a second step, the general pretrained model can be adjusted by adding a final or sometimes several additional task-specific layers to achieve the transfer learning step. Finetuning in general is simply to use the already learned weights of the pretraining step as starting point. It is assumed that these pretrained weights have already learned a rather good representation but can be further exploited by updating them with a smaller learning rate in comparison to pretraining to enhance performance for a specific dataset. We will later see, how models like BERT not only exceed given benchmark scores with finetuning but also learn such powerful representations that they can also be used for feature-based approaches that further process these learned representations and feed them into a new model architecture. In the context of NLP and large Transformer-based language models, representation learning thus gains an additional attraction.

3.1 Evolution of language models

The main framework for a variety of different NLP tasks is the so called language model. A language model estimates a joint probability distribution over a sequence of either words or characters. In the following, the entity of interest will be referred to as tokens. A token can consist of a word, a sub-word or a character, depending on the tokenization algorithm.
For a sequence of tokens $\mathbf{x} = \left(x_1, ..., x_T\right)$, a language model estimates the following probability function:

[3] In the context of the later introduced encoder-decoder architecture, the hidden states of a source sequence x are usually referred to as h while the hidden states of the target sequence y are often denoted by s. This convention is upheld.

$$p(\mathbf{x}) = p\left(x_1, ..., x_T\right) \tag{3.1}$$

This equation is usually factorized into an autoregressive term using the chain rule[4]. In this way, the joint probability of random variables can be factorized into subsequent conditional probabilities. While it is possible to factorize the probability into either a forward or a backward product, the backward variant is often preferred in an NLP context. A language model can then be interpreted as estimating the next token, given all previous tokens in a sequence, id est the left context of the token of interest. This reduces the problem to estimating each conditional factor (Dai et al. 2019):

$$p(\mathbf{x}) = \prod_{t=1}^{T} p\left(x_t|\mathbf{x}_{<t}\right) \tag{3.2}$$

Historically, different approaches have been used to estimate the conditional probabilities of $p\left(x_t|\mathbf{x}_{<t}\right)$ ranging from n-gram modeling (Jurafsky and Martin 2019) to later using neural networks to not only learn the probability function $p(\mathbf{x})$ but also distributed representations of the word vectors at the same time (Bengio et al. 2003). With the implementation of distributed representations of word vectors, it became possible to make use of a rich vector space and comparing words in their similarity thus capturing semantic meaning. This key moment of moving from one-hot encoding to distributed word representations is also crucial, since it allowed for "learned vectors [to] explicitly encode many linguistic regularities and patterns" (Mikolov et al. 2013, p.1). This even goes as far as enabling linear translations on these embedded vector representations that are meaningful[5].

The idea of conditioning the probability of a token on the previously seen sequence of other tokens makes sense intuitively. The probability of a token to occur seems to be greatly influenced by what was already stated. In NLP however, researchers soon leveraged performances by not only conditioning on this so called left but also the right context (id est the tokens that come *after* the token of interest). In the context of representation learning for word vectors, this idea was first exploited by Peters et al., taking traditional unidirectional word embeddings such as word2vec (Mikolov et al. 2013) and GloVe (Pennington, Socher, and Manning 2014) to the next level. For a long time, bidirectionality was only captured in a shallow manner, which means that both undidirectional representations were concatenated together. Bidirectionality plays a central role in understanding the difference between BERT and XLNet, which will be addressed later on.

But not only the notion of bidirectionality but also the idea that a language model should be able to focus on different parts of a sequence that are of more importance for semantic understanding shaped the development of modeling techniques in NLP. In 2014, the idea of sequence to sequence models (seq2seq) that combine two RNNs, namely the encoder and the decoder, was introduced by Sutskever, Vinyals, and Le. While the encoder produces the encoding of the source sequence $\mathbf{x}$, the decoder is simply a language model conditioned on the encoder that generates the target sequence $\mathbf{y}$, sequence

[4]For more details, go to appendix A equation A.1.

[5]Mikolov et al. give the following example: the distributed representation vector of the word [MADRID] - [SPAIN] + [FRANCE] is closer to that of [PARIS] than to any other word representation.

by sequence or token by token for that matter. The problem with this seq2seq architecture is the informational bottleneck. The only context the encoder provides for the decoder comes from the last hidden layer of the encoder, id est the hidden state h_T. The decoder gets only fed a single vector containing all information about the source sentence. For this reason, the attention mechanism was introduced a year later, first in the context of neural machine translation. With the attention mechanism, a decoder can focus on particular parts of the source sentence by employing a direct connection from the encoder to the decoder at each step. This idea was first introduced by Bahdanau, Cho, and Bengio in 2014. The introduced neural machine translation model aims to model the probability of a target sequence $\boldsymbol{y}$ that is factorized as described in equation 3.2 but additionally also takes a context vector c into consideration. The aim is thus to maximize

$$p(\mathbf{y}) = \prod_{t=1}^{T_y} p\left(y_t | \mathbf{y}_{<t}, c\right) \tag{3.3}$$

The key point that Bahdanau, Cho, and Bengio make, is to model the conditional probabilities $p(y_i|\mathbf{y}_{<t}, c)$ such that c is not equal to the last hidden state h_T of a source sequence $\mathbf{x}$. Instead the context vector is defined as c_i and therefore depends on the current target token that the model tries to predict. This context vector c_i is enriched by defining it as the weighted sum of all hidden states of the source sequence $\mathbf{x}$:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{3.4}$$

The scalar values α_{ij} are calculated with a softmax-like function

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{ik}\right)} \tag{3.5}$$

The final and most interesting bit is the definition of e_{ij} which is a scalar that measures how well the input around position j and the output at position i match. It is simply defined as

$$e_{ij} = a(s_{i-1}, h_j) \tag{3.6}$$

where the function a is referred to as *alignment model*. It is not important how exactly a is defined and Bahdanau, Cho, and Bengio propose single-layer perceptron with a tangent activation function. More important is the intuition behind this score and the fact that it considers the previous hidden state of y_i which is denoted s_{i-1} and matches it to the "current" hidden state of x_j, namely h_j[6]. In doing so, "e_{ij} reflects the importance of [...] h_j with respect to the previous hidden state s_{i-1} in deciding the next state s_i and generating y_i" (Bahdanau, Cho, and Bengio 2014, p.4). This mechanism is referred to as encoder-decoder attention. In considering all hidden states of the source sequence $\mathbf{x}$ when translating it to a target sequence $\mathbf{y}$, the problem of the informational bottleneck is solved. However, the computation

[6] In the context of the encoder-decoder architecture, the hidden states of a source sequence x are usually referred to as h while the hidden states of the target sequence y are often denoted by s. This convention is upheld.

of the hidden states are still implemented using recurrent network structures such as LSTMs or GRUs. Another two years later, Vaswani et al. further exploited the idea of encoder-decoder attention by introducing Transformer, a model architecture that allows for unsupervised representation learning using attention mechanisms without recurrence for the first time and making use of the so called self-attention mechanism. Since BERT is largely based on the Transformer architecture, the main design choices are quickly recaptured.

3.2 Transformer

Until the introduction of Transformer, language models were usually either based on some kind of recurrent network, like LSTM or GRU, or on a convolutional network. Common model architectures for language models would usually consist of bidirectional LSTMs used in an encoder-decoder structure. While RNNs in general bear the great advantage of processing sequences of arbitrary length, parallel computation is not possible due to the sequential nature of the input processing. Practically speaking, this means that this kind of sequential modeling is expensive in time and memory resources.

Transformer is also based on an encoder-decoder architecture but one that uses only attention mechanisms *without recurrence* or convolution, which leads to easy parallelization. Transformer consists of 6 identical stacked layers for the encoder as well as the decoder. Each encoder layer consists of two sublayers, namely the multi-headed self-attention layer and a fully connected feed-forward layer. The decoder has a third sublayer which is the enocder-decoder attention layer and uses masking for the self-attention layer.

Broadly speaking, there are two kinds of attention mechanisms in Transformer. The encoder as well as the decoder both use self-attention layers. The decoder additionally uses an encoder-decoder attention which corresponds to the usual encoder-decoder attention mechanism in seq2seq models mentioned earlier on.
Transformers defines every attention mechanism as a function that maps a query and a set of key-value pairs to an output. The dimensions for the queries and keys are always the same while the dimension of the values can be different. For the Transformer however, the query, key and value are all vectors with the same dimension $d_k = d_v = 64$. The output of the attention function is a weighted sum of the value vector. The weight that is assigned to each entry in the value vector is interpreted as measuring the compability of the query with the corresponding key. In practice, Transformer applies the attention function to matrices $Q, K, V \in \mathbb{R}^{T \times d_k}$ that contain sets of query, key and value vectors respectively. The parameter T denotes the (maximal) sequence length.

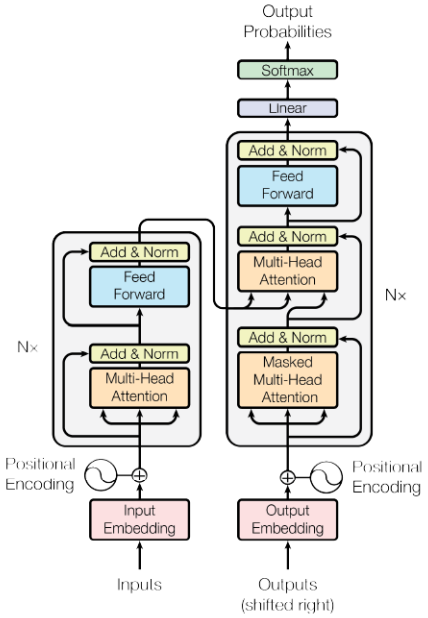

Fig. 3.1: Transformer architecture
Source: Vaswani et al. 2017, p.3.

The self-attention mechanism is then defined as:

$$\text{Attention}\,(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.7}$$

The compability function is the softmax applied to the dot-product of the query and key matrices. The additional scaling with $\frac{1}{\sqrt{d_k}}$ is supposed to stabilize the function and is not part of the usual implementation of the multiplicative attention mechanism.

But what exactly are the query, key and value vector? As already mentioned, there are different variations of the attention mechanism that are used in the Transformer. For the encoder the self-attention function simply takes the same input for the query, key and value vector, namely the output of the previous layer. The first encoder layer takes the embeddings of the token sequence as input. This means that each token can attend to all positions. Using the self-attention mechanism, the different tokens in a sentence are considered in order to obtain a representation of the full sequence.

The decoder also uses self-attention with one important difference: in order to preserve the autoregressive property that other models like LSTMs have, all tokens that come after the current position are masked out by setting them to $-\infty$. The authors explain that this is implemented inside of the compatibility function, id est the softmax function. As a last variant, Transformer also uses an encoder-decoder attention mechanism. In contrast to the self-attention that is supposed to learn something about the representation of an input sequence without considering any corresponding other

Transformer	Layers	Hidden size	Embedding size	Attention heads
Vanilla	6+6	512	512	8

Table 3.1: Original Transformer implementation. There are 6 encoder and 6 decoder layers.

input, the encoder-decoder attention brings a pair of two input sequences together. The query comes from the previous layer, while the key and value come from the encoder and are thus interpreted as "memory" of the model. The current position in the decoder can attend over all position in the input sequence.

This architecture makes sense, when one thinks of the task for which the Transformer was introduced, which is machine translation. Pretraining is then performed on data pairs of source and target sentences that are both embedded and then fed to the encoder and decoder as described and as shown in figure 3.1.

As a last remark about attention, the term multi-head attention is explained. Vaswani et al. found that it was beneficial to not only linearly project the queries, keys and values once but instead h times. These h different linear projections are concatenated and then brought the correct dimension using a final attention weight matrix for each element. The authors state that "multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions" (Vaswani et al. 2017, p.5). The attention thus gets more diverse in a sense. In the Transformer vanilla implementation that is proposed in the paper, 8 attention layers are used, which are referred to as "attention heads".

Attention is the main backbone of Transformer but not the entire architecture. After processing the input inside the attention sublayers, Transformer uses position-wise fully connected feed-forward sublayers with an inner dimension of $d_{ff} = 2048$. This layer is applied to each position separately but uses the same weights (within each sublayer, not across the stacked layers). Its input and output dimension are $d_{hidden} = 512$ as for all attention layers. For an input $\boldsymbol{x}$ the feed-forward layer FF is defined as

$$\mathrm{FF}(\boldsymbol{x}) = \max\left(\mathbf{0}, \boldsymbol{x}\boldsymbol{W_1} + \boldsymbol{b_1}\right)\boldsymbol{W_2} + \boldsymbol{b_2} \tag{3.8}$$

The feed-forward layer thus applies two linear transformations and a ReLU activation[7] (Agarap 2018). The parallelization lends itself from the fact that all processed vectors can be processed in parallel in the feed-forward layer after the application of self-attention .

The embedding of then input sequence is learned during pretraining and converts the input sequence of the source and target sentence to vectors of the dimension $d_{embed} = 512$. The two weight matrices for the encoder and decoder embeddings are shared. Since the model does not contain any explicit

[7]Go to appendix A.5 for more details.

recurrence, it does not know about the positions of the different tokens in the processed sequences. Transformer therefore uses fixed positional encodings that try to incorporate some notion of relative encoding. The positional encoding has the same dimension $d_{embed} = 512$ as the embeddings, since the encoding and embedding are simply summed up for all tokens. The positional encoding PE is defined as

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(pos/10000^{2i/d_{\text{model}}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(pos/10000^{2i/d_{\text{model}}}\right) \end{aligned} \tag{3.9}$$

The encoding of the input vector is equally divided into the first part that is defined by a sine and the second part that is defined by a cosine function. It takes the position *pos* and *i* the dimension of the model, in this case $d_{model} = 512$.

The last brick of the Transformer is the final linear projection of the decoder output and the calculation of the softmax to gain a distribution for the next-token prediction.

In order to stabilize pretraining, Transformer implements usual regularization and stabilization techniques such as dropout (Srivastava et al. 2014), residual connection (He et al. 2015) and layer normalization (Ba, Kiros, and Hinton 2016). Each sublayer has a dropout probabiliy of $p_{drop} = 0.1$. The dropout is applied to the output of each sublayer and is also to the sum of the embeddings and positional encodings. After this, each sublayer uses residual connection and layer normalization which yields to a final calculation in the form of Layernorm(Sublayer($\boldsymbol{x}$) + $\boldsymbol{x}$). Residual connection was originally introduced for image recognition and tackles the problem of model degradation. Model degradation describes the process where a model's accuracy saturates at some point and then degrades rapidly when the network's depth is increased (He et al. 2015, p.1)[8]. Residual connection builds on using the residual function $\mathcal{F}(\boldsymbol{x}) = \mathcal{H}(\boldsymbol{x}) - x$ where the original mapping $\mathcal{F}$ is interpreted as difference between a desired underlying mapping $\mathcal{H}$ and the identity function for $\boldsymbol{x}$. Simply solving the equation for $\mathcal{H}$ results in the residual formulation $\mathcal{H}(\boldsymbol{x}) = \mathcal{F}(\boldsymbol{x}) + \boldsymbol{x}$ that is used in the Transformer. In practice this results in networks that are easier to optimize and have an improved performance due to increased depth (He et al. 2015). Normalization techniques are important to consider for neural networks since the weights in a layer strongly depend on the output of the previous layer. If the output of a previous layer results in large values for the hidden states, the weights that are applied next, have to be larger, too. The idea is to make the weight calculation more independent from this unwanted effect. Layer normalization is independent from the batch size since normalization is done with respect to the hidden units and not the training examples only. Normalizing can reduce the learning rate and make learning more stable (Ba, Kiros, and Hinton 2016, p.4).

[8]The problem of model degradation is also discussed in Lan et al. 2019 where the authors specifically try to mitigate this problem by proposing the ALBERT architecture. Go to chapter 3.6 for more details

In total, the vanilla Transformer consists of 6 stacked layers of encoders and decoders. Both are fed an embedded input with positional encoding of a dimension $d_{embed} = 512$. Through multiple attention heads, the input is projected into a lower-dimensional subspace of dimenion $d_v = 64$ and then reprojected into the 512-dimensional space. After the application of the attention function, the input gets passed through a feed-forward network. The decoder uses a masking strategy to only attend to tokens up to and inluding the current position to preserve a notion of autoregression. In addition, the decoder uses the outputs of the last hidden layer of the encoder stack as well as the output of the previous decoder layer to implement an encoder-decoder mechanism. After passing the input through the two attention layers in the decoder, another feed-forward pass is conducted. Processing the whole input through all 6 layers of the encoder and decoder, a final linear projection is calculated before applying softmax for the next-token prediction. Every sublayer uses residual connection, layer normalization and dropout to either make pretraining faster or more stable.

The Transformer architecture was originally introduced for a machine translation task. The authors have observed that their model also performs surprisingly well on other tasks. This idea of transfer learning is further enhanced by BERT, BERT-based models and XLNet.

3.3 BERT

In October 2018, Google AI published a Transformer based pretrained model that achieved new state of the art results on eleven NLP benchmark tasks at the time of publication. With the introduction of BERT (Devlin et al. 2018) - short for Bidirectional Encoder Representations from Transformers - the possibility of *simultaneously* learning left and right word context was introduced. Up to this point, introducing bidirectionality was only possible by modeling two separate networks for each direction that would later be combined. Furthermore, the idea of creating large pretrained models that can be used to downstream tasks is enhanced. After learning deep bidirectional representations from unlabeled text, BERT can be used for either finetuning or feature extraction. In addition, the model not only excels in performance for sentence-level tasks such as paraphrasing but also for tasks that are on a token level such as named entity recognition. As such, BERT cracked transfer learning for NLP in 2018 and was published in two model sizes for the NLP community (see table 3.2 for more details). The backbone of BERT's architecture is the Transformer.

As a pretraining objective BERT uses a combination of the following two tasks

1. Masked Language Model
2. Next Sentence Prediction

The idea of the Masked Language Model is to randomly mask a specific proportion of all tokens and to train the model to predict these masked tokens. It is inspired by the cloze task, where a person's

language comprehension is challenged by equally having to fill in blanks of removed words from a sentence. In BERT, 15% of the tokens are uniformly chosen for a possible masking during pretraining with the objective of predicting the original vocabulary id based on the unmasked tokens which pose the context. It is the final hidden vector corresponding to the masked tokens that is fed to a softmax function over the vocabulary. The masked token is used to predict the original token by using a cross-entropy loss. The cross-entropy loss measures the performance of a classification model that has a probability distribution as output by using the logarithmic probability that an observed event is classified as $y = 1$ (Hastie, Tibshirani, and Friedman 2009, p.308). The more the predicted probability deviates from the true label, the higher the loss is. In the case of multi-class labeling the cross-entropy loss for an instance i with an input x_i and a given true label y_i is defined as

$$Loss_{CE}(y_i, f(x_i)) = -\sum_{k=1}^{K} y_{k,i} \log \hat{p}(x_{k,i}) \tag{3.10}$$

with K being the number of classes, $y_{k,i}$ a binary indicator if the current input x_i is correctly classified for class k and $\hat{p}(x_{k,i})$ the estimated probability for input x_i to be class k[9].

Only around 80% of all tokens that are chosen for a possible masking are actually masked. This means that the respective token is simply replaced with the special token [MASK]. Another 10% is replaced with a randomly chosen token and the last 10% remain unchanged. This masking procedure was implemented to mitigate the arising discrepancy between pretraining and finetuning since the [MASK] token is purely artificial and can never be observed when finetuning the pretrained model. Using a random token in 10% of the cases did not seem to hurt BERT's language understanding capabilities. The exact ratio of 80/10/10 was found by experimenting with different proportions, also considering to mask all of the chosen tokens with either [MASK] or a random token. Devlin et al. 2018 observe that replacing all 15% with the actual [MASK] token hurts the performance on downstream tasks such as named entity recognition. Likewise, replacing all chosen tokens for potential masking with another random token is reported to worsen performance on a variety of tasks. The implemented masking strategy of replacing 80% with the actual [MASK] token, 10% with another random token and 10% with nothing, id est keeping the original token, was found to yield the best overall performance. Since the model does not know which tokens it will have to predict and which words will be replaced by random tokens, it is forced to keep a distributional context representation of every input token. The masking is applied after tokenizing the data input with a WordPiece model, which will be discussed in chapter 3.3.2. In contrast to a standard language model, as discussed in chapter 3.1, BERT does not predict every token of a sequence but merely the 15% of tokens that are sampled for masking. Likewise, it is not a traditional denoising autoencoder that aims to reconstruct its entire input, because BERT only reconstructs the corrupted input of the masked tokens. Since BERT only uses a comparably small

[9]For a given machine learning problem, a function f(x) is sought that predicts y given the values of the input x, such that a loss function L(y,f(x)) is minimized (Hastie, Tibshirani, and Friedman 2009, p.37).

amount of tokens for prediction to learn its representations, more pretraining steps might be required for the model to converge compared to a standard unidirectional language model. The authors discuss this by stating that the performance improvements outweigh the increased training cost by far. In this case, the authors are certainly right. However, other model architectures (DistilBERT and ALBERT notably) will go deeper into discussing the trade-off between training costs and enhanced performance.

The Next Sentence Prediction (NSP)[10] task is supposed to allow BERT to understand relations between different segments of sequences which is not captured directly by language modeling itself. The NSP task was specifically introduced to improve performance on NLP tasks such as question answering and natural language inference that require understanding the relationship between segments. The next-sentence prediction is a binary task that predicts whether or not segment B follows segment A. This means that each training example consists of a text pair (segment A, segment B). The starting point is always the first segment A. In 50% of the cases, the second segment B is the actual segment that follows segment A. While in the other 50% of the cases, BERT randomly selects a segment from the whole corpus. The cases are then labeled *IsNext* and *NotNext* respectively. The two segments are concatenated and fed as one input sequence to the model. They are distinguished by the special separation token [SEP]. In addition to the [SEP] token, all input examples are embedded starting with a classification token [CLS]. During pretraining, the last hidden representation of this [CLS] token is used for making the next sentence prediction. An input example thus takes the form [CLS] segment A [SEP] segment B [SEP]. The following two inputs are examples for the *IsNext* and *NotNext* categories:

[CLS] THE MAN WENT TO [MASK] STORE [SEP] HE BOUGHT A GALLON [MASK] MILK [SEP]
[CLS] THE MAN WENT TO [MASK] STORE [SEP] PENGUIN [MASK] ARE FLIGHT LESS BIRDS [SEP]

As the examples indicates, the masking is performed on the two segments individually. The hashes are introduced by the WordPiece model that BERT uses to tokenize its input and breaks two words into sub-word units.

It was already mentioned that all models are closely related to the introduced Transformer architecture. The biggest difference to the vanilla implementation of Transformer is that BERT only uses stacks of encoders. BERT base uses 12 instead of six encoder layers. In addition, BERT uses a larger embedding size of 768 instead of 512. See table 3.2 for more details on the available model sizes for BERT. Both Transformer and BERT rely on attention mechanisms. While the Transformer uses self-attention, masked self-attention and encoder-decoder attention, BERT only uses self-attention. It is important to understand that the two masking strategies have different goals and have to be seen as

[10]BERT uses the term "sentence" instead of "segment". In the paper "sentence" is defined as an arbitrary span of text and not as a sentence in a linguistic sense. In this context an arbitrary span of text will be referred to as either "segment" or "sequence" to avoid confusion.

BERT	Layers	Hidden size	Embedding size	Attention heads
base	12	768	768	12
large	24	1024	1024	16

Table 3.2: Available BERT versions

different approaches. The Transformer uses masking in the decoder to introduce some mechanism of autoregression. When translating a source to a target sentence, the model is supposed to only refer to the already seen positions. The masked tokens are simply set to $-\infty$. In BERT, masking means that a token is actually replaced with the token [MASK], randomly replaced by another token or remaining as it is. BERT is trained on predicting the [MASK] tokens while Transformer simply processes its input differently when masking. Overall, BERT is simply a stack of the encoder side of the Transformer, implementing all main architecture choices such as dropout, residual connection, layer normalization and of course the attention mechanism that were already described in chapter 3.2.

3.3.1 Pretraining

BERT is pretrained on the BooksCorpus with 800 million words as well as the English Wikipedia text passages of 2,500 million words without lists, tables and headers. The authors specifically mention the importance of using a document-level corpus to extract long contiguous sequences. The data is first tokenized using a WordPiece model[11] (Wu et al. 2016). After that, the masking procedure is applied for each instance in the corpus. Yinhan Liu et al. report that BERT used a duplicated version of its dataset and masked the tokens for each of these duplicated datasets, for more details on this procedure refer to chapter 3.4. The combined length of the input sequences with all additional special tokens is set to ≤ 512 which means that the sequence length is restricted to 512 tokens. Since longer sequences are disproportionately expensive, the authors further reduced computation cost by only using 128 tokens as sequence length for 90% of the pretraining steps. For the remaining 10% of the steps, the "full" sequence length of 512 tokens was used to learn the positional embeddings. The authors provide no analysis as to how much this hurts performance. BERT is pretrained with a batch size of 256 which results in around 128,000 tokens/batch since the sequence length is set to 512.This indicates that using a sequence length of 128 was achieved by padding the 384 remaining positions. As optimizer, Adam[12] (Kingma and Ba 2014) is used with a learning rate of 1e-4, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. In addition an L2 weight decay of 0.01, a learning rate warm up over the first 10,000 steps and a linear decay of the learning rate are used. As activation function, GELU (Gaussian Error Linear Units)[13] (Hendrycks and Gimpel 2016) is implemented which is the same as for GPT and all layers use a dropout probability of 0.1. For BERT base, 4 Cloud TPUs and for BERT large 16 Cloud TPUs were used for training which resulted in 16 and 64 TPU chips respectively. The authors state that each model took 4 days to train.

[11]The WordPiece model is explained in more detail in chapter 3.3.2.
[12]Go to appendix A.1 for more details on the Adam algorithm.
[13]Go to appendix A.4 for more details.

3.3.2 Tokenization and embedding

For pretraining, BERT uses WordPiece embeddings and has a vocabulary size of around 30,000 tokens. The exact size depends on the pretrained version that is used for BERT. Using a BERT model that was trained on cased text, the vocabulary size is 28,996 while using a model trained on uncased text, the vocabulary consists of 30,522 tokens. Out of all considered models, BERT is the only one that is available for cased and uncased pretraining. For the analysis of the Fake News dataset, the cased model was used, see chapter 6.1 for more details. The WordPiece model looks at sub-word units instead of whole words. Since words are divided into a limited set of common sub-words, also called "wordpieces", the handling of rare words get easier (Wu et al. 2016, p.1). Using the WordPiece model, BERT finds a balance between models that are based on characters and very flexible and models that look at whole words which is more efficient. In order to separate whole words from sub-word units, the latter are indicated by starting them with hashes. This enables the model to recover a sequence without ambiguity (Wu et al. 2016, p.7). As already mentioned, every sequence starts with a special [CLS] token for classification. When performing a classification task for finetuning BERT, the only token of which the hidden state is further processed by the additional finetuning layers is this [CLS] token. Since the token is already used for the NSP pretraining objective, the authors state that it only becomes a useful summary of the sequence representation after finetuning (Devlin et al. 2018, p.4). In addition to the token embedding, BERT also learns a binary segment embedding that simply checks to which segment each token belongs as well as a positional embedding that marks the absolute position of each token within the length of up to 512 tokens. As already mentioned this positional embedding is learned by only using 10% of the total pretraining steps. All embedding matrices are updated during finetuning if not otherwise specified. During finetuning BERT is able to handle a single sequence input as well as a pair of segments.

3.4 RoBERTa

The main premise of RoBERTa (Robustly optimized BERT approach) (Yinhan Liu et al. 2019) is the assumption that BERT was seriously undertrained during pretraining. RoBERTa is thus trained with bigger batches over more data for a longer time on longer sequences. In addition, Yinhan Liu et al. argue that the second task of the next-sentence prediction doesn't improve BERT's performance in a mentionable way and therefore remove the task from the training objective. Furthermore, the authors introduce a dynamic masking pattern. The improvements are therefore twofold: on one hand, pretraining was simply done more excessively and on the other hand several architectural changes were made. With a stack of 12 encoder layers, a hidden size of 768 and 12 attention heads, the main architecture for RoBERTA base is the same as for BERT base. Like BERT, RoBERTa is available in this base or in a larger version with a doubled size of encoder layers, a bigger hidden size and more attention heads as indicated in table 3.3.

For BERT, masking was done in a static way. This means that masking was only performed once for each training instance during data preprocessing. In order to maintain more variety, BERT then duplicated the whole training dataset 10 times, to have ten different maskings for each training instance. Since BERT was trained for around 40 epochs (Devlin et al. 2018, p.13), each training instance was seen four times during training. In contrast to this approach, Yinhan Liu et al. introduce a dynamic masking pattern which conducts the masking every time an instance is fed to the model and not only once during data preprocessing. This yields comparable results to static masking while simultaneously being more efficient since it is not necessary to duplicate the training set for dynamic masking.
Although Devlin et al. have reported serious performance loss when removing the NSP, the necessity of this additional task was later questioned. Yinhan Liu et al. experiment with different kinds of inputs while keeping a fixed batch size of 256 and using the datasets of BERT's pretraining as well as the BERT architecture. They compare four different input versions. The first keeps the structure that BERT proposes and uses segment pairs. The second also uses a pair of inputs but goes down on a sentence level. In this case, this means that sentences in a linguistic sense are used. As third option, they explore using full sentences that can originate from different documents. While the last version is the same but uses full sentences that come from the same document only. All inputs are restricted to a maximum sequence length of 512. While the first two setups still retain the NSP loss, it is removed for the latter two. The experiment shows that going down on a sentence-pair level hurts the performance. Interestingly, removing the NSP loss itself doesn't lead to a general performance loss. This contradicts the experiments that Devlin et al. have done themselves. Yinhan Liu et al. hypothesize that the authors of BERT might have simply removed the NSP task while still retaining the segment pair input structure, while for RoBERTa this task was not only removed but the input was changed to a concatenated version of the two segments without a [SEP] token between them by default. In order to be able to better compare RoBERTa's architecture to other works, the authors proceed with the full sentence input from different documents in pretraining. The comparison is then easier because this procedure results in a steady batch size. This stems from the fact that for using full sentences from different documents, sentences from the next document were sampled once the end of a document was reached. To indicate that there were different documents involved, they were split by using a [SEP] token between them. For the full sentences that were sampled from the same document only, input sequences could be shorter than 512 tokens when reaching the end of a document. The batch size was increased in these cases to reach a similar tokens per batch ratio as for the other input versions.

3.4.1 Pretraining

RoBERTa can be seen as the result of a replication of BERT pretraining that specifically evaluated the choice of hyperparameters as well as training set size during pretraining. In total, RoBERTa's architecture differs from BERT in training the model longer with bigger batches over more data. In addition to the BooksCorpus and English Wikipedia data that BERT uses, RoBERTa is pretrained on three more datasets. The first additional dataset is called CommonCrawl News dataset and contains 63

RoBERTa	Layers	Hidden size	Embedding size	Attention heads
base	12	768	768	12
large	24	1024	1024	16

Table 3.3: Available RoBERTa versions

million news articles from September 2016 to Feburary 2019 in English. OpenWebText is a dataset with web content and was extracted from URLs that were shared on Reddit and received at least three upvotes. Finally, the dataset Stories was used which incorporates a more story-like writing style. In contrast to BERT, no duplication of the dataset was necessary since a dynamic masking strategy was used.

In general, training with larger batches increases the performance of a network and trains faster (Goodfellow, Bengio, and Courville 2016, p.276). Increasing the batch size needs to be done in adjustment with a higher learning rate. Another advantage of an increased batch size lies in the better utilization of available hardware since smaller batch sizes are harder to parallelize. For RoBERTa, the batch size is increased to 8,000 during pretraining. BERT originally uses a batch size of only 256 but the authors give no indication as to why they use such a small batch size. The authors might have simply not examined the hyperparameter batch size for pretraining, although the importance of the batch size for the performance is well known and will be examined for finetuning in chapter 6.4. There have been experiments of using a batch size of up to 32,000 for BERT, although they have relied on a different optimization algorithm (You, J. Li, et al. 2019). The learning rate is increased from lr = 1e-4 for BERT (base and large) to lr = 6e-4 for RoBERTa base and lr = 4e-4 for RoBERTa large. RoBERTa uses a linear learning rate decay like BERT but with more warm up steps. BERT uses 10,000 warm up steps (for base and large) while RoBERTa uses 30,000 warm up steps for RoBERTa large and 24,000 warm up steps for RoBERTa base. In contrast to BERT, RoBERTa is only trained on full-length sentences for all steps. The maximum sequence length is equally set to 512 tokens. The optimization algorithm is also Adam[14], with $\beta_1 = 0.9$ as in BERT and $\beta_2 = 0.98$ which is slightly lower than for BERT which used $\beta_2 = 0.99$. Yinhan Liu et al. found this lower value for β_2 to be improving stability when increasing the batch size. RoBERTa was trained on DGX-1 machines with 8 x 32GB Nvidia V100 GPUs.

3.4.2 Tokenization and embedding

Like BERT, RoBERTa uses an embedding architecture that relies on subword units although the used tokenization algorithm is different. Byte-pair encoding is used on a byte-level and was introduced by GPT-2 (Radford et al. 2019). By using bytes instead of unicode characters, a moderate vocabulary size of 50,000 can be learned that can encode a corpus without introducing any [UNK] tokens. Even though, the authors of RoBERTa report a slightly worse performance when using this byte-level encoding, they chose it in light of the advantages of a universal encoding scheme. Since the vocabulary

[14]Go to appendix A.1 for more details on the Adam algorithm.

size is much bigger than for BERT, more parameters are used due to the bigger embedding matrix. For RoBERTa base, the total number of parameters thus increases from 110 million to 125 million.

3.5 DistilBERT

While RoBERTa focuses on exploiting pretraining by scaling up the use of data, batch size and training time, DistilBERT (Distilled BERT) (Sanh et al. 2019) was introduced as a light version of BERT that might not excel as well with respect to certain performance measures but is actually usable under constrained computational training and inference budgets. Sanh et al. criticize the idea of simply enlarging datasets and run more and more exhaustive pretrainings since it doesn't consider computational costs, memory requirements and even environmental aspects (Hao 2019, T. Peng 2019) that are often neglected for the sake of further enhancing performance. DistilBERT thus critically explores the size of the BERT architecture and ultimately ends up radically diminishing it by around 40% while retaining 97% of its language understanding capabilities and being 60% faster. This is mainly achieved by reducing layers. Since Sanh et al. use a distinct knowledge distillation approach, the pretraining loss for DistilBERT is changed. In contrast to RoBERTa, changing the pretraining objective is not justified by questioning the general purpose of BERT's pretraining loss.

Wanting to distill a large architecture can make extensive research on the use and purpose of layers and building blocks necessary. Indeed, understanding the necessity and effect of BERT's different building blocks is such a big research area that the NLP community refers to this with the term "BERTology". While DistilBERT is not the first attempt to use BERT's large architecture and scale it down, in contrast to other approaches it tries to keep the main Transformer architecture as much as possible (Tang et al. 2019). Sanh et al. simply remove the segment embeddings that BERT uses to indicate if a token belongs to segment A or B and that is added to the positional and token embedding. In addition, the authors refer to the pooler layer being removed during pretraining. This pooler layer is reintroduced during finetuning in the implementation the authors have made available[15]. Its removal therefore mainly results in a different initialization which is random for DistilBERT while BERT uses the pretrained weights as initialization. The biggest reduction comes from only using one out of two encoder layers which results in stacking only 6 encoder layers instead of 12 for the base version. Other techniques used in BERT such as a layer normalization and dropout are retained since they are implemented in a highly efficient manner and important for a stable training. The authors have also observed that changing parameters such as the dimension of the hidden size have less impact on the computational efficiency than simply reducing layers.

[15]This concerns the DistilBertForSequenceClassification model that is made available by Huggingface (Wolf et al. 2019) for classification tasks which was used for the analysis of the Fake News data. For further details on the implementation, go to chapter 6.1.

DistilBERT	Layers	Hidden size	Embedding size	Attention heads
base	6	768	768	12

Table 3.4: Available DistilBERT version

As already mentioned, DistilBERT is trained on a different objective than BERT. This comes from the compression technique *knowledge distillation* that Sanh et al. are using. Using *knowledge distillation*, a small model is trained such that the behavior of a large model is reproduced. This is achieved by introducing an additional distillation loss that focuses on aligning the estimated distribution of a student (the small model, in this case DistilBERT) and a teacher (the large model, in this case BERT). The Masked Language Model loss is extended by this distillation loss which consists of two parts. The first task that BERT is trained on is the Masked Language Model task that results in minimizing the cross-entropy loss between the model's predicted distribution over the vocabulary size and the actual token that is covered by the [MASK] token. The model thus minimizes the cross entropy between the predicted probability of the model and the empirical distribution. A good model predicts a high probability for the correct class and near-zero probabilities for the rest. Sanh et al. argue that these near-zero probabilities contain additional information of value. Since they still differ in how close they are to zero, it is part of what the model learns about the structure of a language. These near-zero probabilities therefore reflect the generalization capability of a model. The sentence "I think this is the beginning of a beautiful [MASK]" has high probabilities for the tokens [DAY] and [LIFE] in a BERT base model. The many other valid options are not considered but still of interest, since learning small differences in these predictions is what makes a model like BERT so powerful.
The idea for the distillation loss is to exactly leverage this full teacher distribution of BERT. This loss focuses on aligning the distribution between the student and the teacher which results in a special distillation loss

$$Loss_{distil} = \sum_i t_i \log s_i \tag{3.11}$$

with t_i being the predicted probability of the teacher and s_i the one of the student for an instance i[16]. Both probabilities are gained using softmax with an additional temperature parameter M, thus

$$p_i = \frac{\exp(z_i/M)}{\sum_j (z_j/M)} \tag{3.12}$$

with M being the same value for both student and teacher during pretraining and 1 during finetuning to gain the usual softmax output. For $M > 1$ the probability distribution gets softer (Hinton, Vinyals,

[16]Sanh et al. explain they use a cross-entropy loss. However, the definition that is stated in the paper and given in equation 3.11 does not equate to the cross-entropy loss which would be defined as $Loss_{CE}(t_i, s_i) = -\sum_i t_i \log s_i$. It is assumed that the authors neglect the sign in their definition of $Loss_{distil}$ but calculate the overall loss as $Loss_{overall} = Loss_{MLM}$ - $Loss_{distil} + L_{cos}$ which is equal to calculating the overall loss as $Loss_{overall} = Loss_{MLM} + Loss_{CE} + L_{cos}$. The cosine embedding loss L_{cos} is introduced further down while $Loss_{MLM}$ corresponds to the masked language model loss that BERT already uses.

and Dean 2015, p.3). M thus controls the smoothness of the function, while z_i is the calculated model score for class i. So knowledge transfer between the teacher and the student takes place by training the outputs of the student classifier on those of the teacher. These outputs of the teacher are provided as soft target probabilities which are controlled by the temperature parameter M. In addition to the masked language model loss of RoBERTa respectively BERT and the distillation loss, a third loss, namely the cosine embedding loss $Loss_{cos}$ is used. This latter loss provokes an alignment between the directions of the student's and the teacher's hidden states. Similarity between the two vectors h_t and h_s is defined as the cosine of the angle between the vectors, which results in (Goldberg and Hirst 2017, p.136):

$$\text{sim}_{\cos}(h_t, h_s) = \frac{h_t \cdot h_s}{\|h_t\|_2 \|h_s\|_2} \tag{3.13}$$

The cosine embedding loss is then defined as

$$Loss_{cos} = -\text{sim}_{\cos}(h_t, h_s) \tag{3.14}$$

which is the negative similarity, since a loss is always seeked to be minimized.

3.5.1 Pretraining, tokenization and embedding

Since DistilBERT simply reduces the architecture of BERT, other building blocks such as tokenization, embedding and used data for pretraining remain the same except for the already mentioned changes. DistilBERT is available as a distilled version from BERT as well as RoBERTa.

3.6 ALBERT

ALBERT (A Lite BERT) (Lan et al. 2019) is another model questioning the need for larger and larger pretrained model architectures that might not be usable in real life applications due to memory and time costs. Lan et al. therefore examine the trade off of having better performing NLP models for a large variety of tasks versus having a leaner model that yields results in a faster and sufficiently good way. They specifically point out that not only time resources but also hardware resources are limited. Furthermore, the idea of simply enlarging model architectures by using bigger hidden size dimensions or more layers can lead to unexpected model degradation. With ALBERT, a bert-based architecture is introduced that uses parameter reduction techniques (factorized embedding parameterization and cross-layer parameter sharing) while exceeding the performance of BERT, RoBERTa and XLNet on a variety of benchmark NLP tasks. Taking a closer look at this presented claim of introducing a sufficiently good and faster model, it becomes clear that Lan et al. cannot fulfill this idea. The approach of introducing a lighter model architecture is used in order to scale the model up again, which will be discussed and explained in more detail in the following passages. In total, the authors introduce six changes to the original BERT architecure: the already mentioned parameter sharing, the factorization of the embedding matrix, the use of a different optimizer, the replacement of the NSP with a

segment-coherence loss, the expansion to n-gram masking, as well as the use of SentencePiece instead of WordPiece as a tokenizer.

In all architectures considered so far, the hidden layer size was tied to the embedding size of the simultaneously learned distributed representations. Lan et al. argue however that this poses an inherent contradiction since embeddings are supposed to be context-*independent* representations while hidden-layer embeddings should learn context-*dependent* representations. These two sizes should thus be treated more independently. One can think of the embedding matrix as simply indicating a general distributed representation that only considers the token itself. Passing these general embeddings through the encoder layers and applying the attention mechanism then leads to a context-sensitive representation that also considers the other tokens of the current instance. The ALBERT architecture specifically models this idea by projecting the one-hot vectors that indicate the current token of interest, into a lower-dimensional embedding space and then project these representations to the hidden space. Since this general embedding is not intended to model the precise meaning of a token, it seems plausible to represent them with a reduced dimension. The main advantage however lies in the reduction of parameters. The embedding dimension of ALBERT has a size of $d_{embed} = 128$ instead of the one of BERT of $d_{embed} = 768$. Since BERT has a vocabulary size of around 30,000 tokens, around 23,04 million parameters for the embedding table alone are necessary. While ALBERT, with a vocabulary size of 32,000 tokens and the aforementioned embedding size of 128 only needs around 3,84 million parameters. Since for BERT the embedding size is the same as the hidden layer size, the distributed representations of the embedding table can be simply fed to the lowest encoder layer. For ALBERT, an additional projection matrix is implemented that projects these shortened embeddings from a size of $d_{embed} = 128$ to the hidden size of $d_{hidden} = 768$ that is used for BERT and ALBERT alike. This additional projection matrix adds another $128x768 = 98,304$ parameters to be estimated. In total, ALBERT therefore needs around 3,85 million embedding parameters which corresponds to around 16.7% of the 23,04 million parameters that BERT needs.

There is a second parameter reduction technique that ALBERT implements. In ALBERT, all parameters in the encoder layers are shared. This seems surprisingly simple. BERT is known for being able to even understand finegrained nuances of language and a lot of research is done in understanding the black box of the encoder stack better (Clark et al. 2019, Michel, Levy, and Neubig 2019, Tenney, Das, and Pavlick 2019). As usual, the introduced model architecture of ALBERT is available in different sizes. While for BERT the main distinction is between base and large, ALBERT is also available in xlarge and xxlarge. Both base and xxlarge consist of an encoder stack of 12 layers but the xxlarge version scales the hidden dimension from 768 up to 4,096 and uses 64 instead of 12 attention heads. The xlarge variant is the counterpart to the large version, both of which have 24 encoder layers and hidden sizes of 2048 and 1024 respectively. The only versions of ALBERT that outperform BERT are the ones that use an enlarged hidden size compared to the 768 of BERT. This is true for the xlarge and xxlarge versions only. This is not surprising, since it is implausible that simply sharing parameters

across layers would not hurt performance on downstream tasks. This enlarged hidden size however increases the overall parameters of the model. When only considering the *unique* number of parameters, ALBERT can be considered "lite" in contrast to BERT. But when actually evaluating ALBERT model variants on the usual benchmarks, ALBERT only outperforms BERT when it is scaled up again by implementing a bigger hidden size, thus enlarging the width of the network. The idea and wish for scaling up an existing network architecture is very popular in the NLP community. The authors have observed that simply scaling up the BERT architecture to an xlarge version with 24 layers (the same as BERT large) and a hidden and embedding dimension of 2048 (double the size of BERT large) doesn't further enhance performance on downstream tasks but on the contrary leads to model degradation, id est a worse performance. By reducing the unique parameters and specifically reducing parameters for embedding, an enlarged hidden dimension can be introduced that results in a better performance on NLP benchmarks. Whether this approach is worth the effort, is to be evaluated.

Another important change was conducted by adjusting the training objective of BERT. Yinhan Liu et al. already discussed the ineffectiveness of BERTs NSP task during pretraining. Lan et al. confirm this observation and additionally assume the cause of this ineffectiveness to lie in the lack of difficulty this binary task poses as opposed to the much more difficult Masked Language Model task. The authors argue that BERT's NSP loss ultimately reduces itself to a topic modeling task. A segment pair that does not consist of two consecutive sentences is produced by simply assigning a segment from another document and that means from a completely different context and potentially topic. In contrast to that, positive examples are created by taking two consecutive sentences of the same document. In that case, BERT can simply check if the two sentences use similar words and cover a similar topic in order to predict the pair as a positive example. In chapter 3.3 two example sentences for both cases are given. In contrast to RoBERTa, ALBERT does not completely eradicate this second loss but changes it to a so called inter-sentence coherence loss. While BERT uses two consecutive sentences as positive and two randomly assigned sentences as negative examples for the *IsNext* prediction, ALBERT instead simply changes the order of the positive examples to generate the negative ones arguing that this would force the model learn finer-grained distinctions about discourse-level coherence properties. This makes sense, since both sentences are still related to one another but the order of their appearance is swapped. The task thus becomes more challenging and beneficial for downstream tasks.

In addition to changing the NSP loss, ALBERT also uses a variant of the Masked Language Modeling task by switching to n-gram masking. The length of each n-gram mask for a sequence $\mathbf{x} = \left(x_1, ..., x_T\right)$ is randomly selected with a maximum of n = 3 and probability length of

$$p(n) = \frac{1/n}{\sum_{i=1}^{T} 1/i} \tag{3.15}$$

This means that depending on the sequence length T, the highest probability is assigned to uni-grams, followed by bi- and then 3-grams. This approach was first introduced by Joshi et al. who named their new model SpanBERT. The main reasoning behind masking spans rather than individual tokens is that

it poses a more difficult task and enhances the performance on downstream tasks that require some sort of reasoning such as question answering. It is much harder to correctly predict [NEW] [YORK] if both tokens are masked simultaneously, rather than learning to predict [NEW] as a token, if [YORK] is unmasked. Since for ALBERT the n-grams are randomly selected, they can strike "naturally" belonging n-grams such as [NEW] [YORK] within a sentence. The masking can be applied to any random n-gram of a given sequence, such as [AND] [I] in the sequence "I HATE NEW YORK AND I LOVE PARIS.". Since there is no embedding of tokens somehow belonging together, this approach does not fully solve the problem that XLNet discussed with the implicit independence assumption that BERT uses in its masked language modeling task, see chapter 3.7 for more details.

3.6.1 Pretraining

ALBERT is trained on the same corpus as BERT, namely BooksCorpus and English Wikipedia and uses a batch size of 4,096. As optimizer, LAMB (Layer-wise Adaptive Moments optimizer for Batch training) (You, J. Li, et al. 2019) was chosen with a learning rate of 0.00176. The main goal of LAMB is to handle training with very large batch sizes in order to speed up training. It was first introduced by You, J. Li, et al. to specifically speed up pretraining for BERT which could be reduced from 3 days to 76 hours. LAMB builds on the optimization algorithm LARS (Layer-wise Adapative Rate Scaling) (You, Gitman, and Ginsburg 2017) that is popular among the computer vision community but does not perform well on attention-based models like BERT. Both algorithms depend on the so called *trust ratio* which builds the ratio between the norm of the weights $\boldsymbol{\theta}$ in each layer l and the norm of the gradients $\boldsymbol{g}$. The trust ratio is used to increase the global learning rate ϵ if it is large and vice versa. This is done layer-wise therefore resulting in a layer specific learning rate λ^l:

$$\lambda^l = \epsilon \frac{\left\|\boldsymbol{\theta}^l\right\|}{\left\|\boldsymbol{g}^l\right\|} \tag{3.16}$$

Ultimately, training with large batch sizes is stabilized. LAMB implements the trust ratio by setting the numerator or denominator to 1 if either one of them is 0 and uses the Adam update rule[17], instead of the Stochastic Gradient Descent update rule[18] that LARS is implementing.

All models were trained for 125,000 steps on a Cloud TPU V3 by using 64 to 1024 TPUs. As already mentioned, ALBERT is available in four versions, see table 3.5 for more details.

3.6.2 Tokenization and embedding

The structure of the input follows the same rules as for BERT, which means that ALBERT expects an input of the format [CLS] SEGMENT A [SEP] SEGMENT B [SEP]. While the input format stays the same and ALBERT can also process inputs of two segments, the preprocessing steps for pretraining

[17]The Adam update rule can be found in the appendix A.1 under algorithm 1.
[18]The Stochastic Gradient Descent update rule is defined in the appendix in equation A.6

ALBERT	Layers	Hidden size	Embedding size	Attention heads
base	12	768	128	12
large	24	1024	128	16
xlarge	24	2048	128	16
xxlarge	12	4096	128	64

Table 3.5: Available ALBERT versions

is different. ALBERT uses SentencePiece tokenization (Kudo and Richardson 2018) which has the primary benefit of being applicable independent from language since it processes raw text. It indicates the starting of new words with an underscore. WordPiece models need a clean dataset with text that split sentences into tokens first, based on whitespace and punctuation. This additional step of splitting the tokens is language dependent. SentencePiece gets rid of this.

3.7 XLNet

While RoBERTa, DistilBERT and ALBERT focus on either improving or distilling BERT, the main architecture of using bidirectional encoders remains the same. This is because all discussed BERT-based models rely on the Masked Language Model training objective. With XLNet, Z. Yang et al. propose an alternative approach that is based on modeling language in an autoregressive manner as introduced in chapter 3.1. The paper strongly focuses on discussing the two approaches of denoising encoders and autoregressive language models. Since BERT's objective is to reconstruct a corrupted input, it can be described as a denoising encoder approach. In contrast to the denoising encoder, autoregressive language modeling uses a sequential approach that can only condition on either left or right context. XLNet combines the advantages of both approaches, namely the bidirectionality while capturing dependency structures among tokens better and not suffering from a pretraining-finetuning discrepancy.

The objective of an autoregressive language model is to estimate the probability distribution by maximizing the probability of a given sequence $\mathbf{x} = \left(x_1, ..., x_T\right)$. In practice, this joint probability is factorized into a product of conditional probabilities which results in either considering the left or the right context but never the bidirectional context at the same time. In contrast to this, the autoencoding approach does not perform explicit density estimation for a sequence since its aim is to reconstruct the original data from the corrupted input that is generated with the masking procedure. As the authors of BERT have already debated, BERT suffers from a pretraining-finetuning discrepancy due to introducing the artificial [MASK] token that is only used while pretraining but is never actually present in data during finetuning. To reduce this discrepancy, a masking ratio of 80/10/10 was implemented, where only 80% of the tokens chosen for masking are actually masked, 10% are overwritten with a random token and another 10% remain as they are. Go to chapter 3.3 for more

explanations on this procedure. While this procedure mitigates the effect of the discrepancy, Z. Yang et al. criticize that it doesn't fully eliminate it.

As a second problem, the encoding approach of BERT implies an implausible independence assumption between the masked tokens. BERT masks several tokens in a given sequence at once and tries to reconstruct all masked tokens at the same time. This implies an independence assumption. One can think of the following example:

I WENT TO [MASK] [MASK] AND SAW THE [MASK] [MASK] [MASK]

Potential plausible solutions for this corrupted input could be:
I WENT TO [NEW] [YORK] AND SAW THE [EMPIRE] [STATE] [BUILDING]
I WENT TO [SAN] [FRANCISCO] AND SAW THE [GOLDEN] [GATE] [BRIDGE]

While the following solutions would not be plausible:
I WENT TO [SAN] [FRANCISCO] AND SAW THE [EMPIRE] [STATE] [BUILDING]
I WENT TO [SAN] [YORK] AND SAW THE [GOLDEN] [STATE] [BUILDING]
...

BERT does not distinguish between these plausible and implausible solutions. These kind of dependencies among masked tokens are therefore not learned during pretraining.

XLNet takes the advantages of autoregressive and encoding approaches to combine them into one model. The main idea is to have an autoregressive model that considers bidirectional context. Instead of using a fixed forward or backward factorization of the conditional probabilities, XLNet maximizes the expected logarithmic likelihood of all sequences with respect to all possible permutations of the factorization order. In doing so, the context of each token can consist of tokens from the left and right which makes the model learn to utilize the contextual information from all tokens in expectation. Since the product rule for factorizing the joint sequence probability holds universally, it can be applied.
To understand how XLNet processes an input sequence, consider the following example of a sequence $\mathbf{x} = \left(x_1, x_2, x_3, x_4, x_5\right)$ with the five tokens (I, LIKE, CATS, AND, DOGS). For $\mathbf{x}$, let Z_5 denote the set of all possible permutations of [1,2,3,4,5] for a sequence of length $T = 5$. For this factorization order a random permutation z is sampled, for example [4,2,5,3,1]. A traditional language model would seek to model each token of the original order [1,2,3,4,5] successively. This means a language model using the left context would first predict token x_1, then x_2, x_3 and so forth, always conditioned on the already predicted tokens to the left of the current token. Using the product rule the probability of token x_3 would thus be

$$p(x_3) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1) \tag{3.17}$$

which means that only the left context of token x_3 is considered.

Now consider the permutation language model (PLM) wants to predict token 3 of the original sequence, id est x_3. Using the product rule and the sampled random permutation order [4,2,5,3,1], the probability then becomes

$$p(x_3) = p(x_4)p(x_2|x_4)p(x_5|x_4, x_2)p(x_3|x_4, x_2, x_5) \tag{3.18}$$

For predicting the token x_3, the context from the left (x_2) as well as context from the right (x_4 and x_5) is considered. The probability of predicting $p(x_3)$ = [CATS] therefore relies on the conditional probabilities of the tokens [AND], [LIKE] and [DOGS]. It is important to note that the actual ordering of the sequence remains the same and that permutation is only done with respect to the factorization order z.
Another example further emphasizes the difference to BERT[19]). Assume the sequence (NEW, YORK, IS, A, CITY) has the two tokens [NEW] and [YORK] that are supposed to be predicted. This means that the log p(NEW, YORK|IS, A, CITY) has to be maximized. For XLNet, the assumed factorization order is $z = [3, 4, 5, 1, 2]$. For BERT the corrupted input takes the form ([MASK], [MASK], IS, A, CITY). The following objectives are reduced for the two models BERT[20] and XLNet:

$$\begin{aligned} \mathcal{J}_{\text{BERT}} &= \log p(\text{NEW}|\text{IS, A, CITY}) + \log p(\text{YORK}|\text{IS, A, CITY}) \\ \mathcal{J}_{\text{XLNet}} &= \log p(\text{NEW}|\text{IS, A, CITY}) + \log p(\text{YORK}|\textbf{NEW}\text{, IS, A, CITY}) \end{aligned} \tag{3.19}$$

The autoregressive formulation allows XLNet to capture the dependency between [NEW] and [YORK] while simultaneously considering the bidirectional context.

Implementing this new approach of autoregression with a bidirectional context consideration, an additional complication arises. The final layer uses the product of the embedding of the original sequence with the hidden layer of the permuted sequence in a softmax to produce the next-token probability. In order to predict the token x_{z_t} the context of the previous tokens is used. In the case of XLNet this corresponds to the permuted context sequence $\mathbf{x}_{\mathbf{z}<t}$. The context is used by considering the last hidden representation $h(\mathbf{x}_{\mathbf{z}<t})$ of it. This hidden representation ignores the target position of the token target $\mathbf{x}_{\mathbf{z}_t}$. For every token, the same distribution over the vocabulary size is predicted. Hence, there is a need for a hidden layer representation that considers the bidirectional context as well as the target position. In XLNet this is achieved by using a two stream attention mechanism that consists of a content and a query representation. For a given layer m, the content representation $h_{zt}^{(m)}$ encodes the context as well

[19]The example is taken from the XLNet paper (Z. Yang et al. 2019, p.6
[20]For demonstration purposes the NSP loss is not considered.

as the current token itself. The query stream $g_{z_t}^{(m)}$ encodes the context and the current position. Both representations are updated with a self-attention mechanism as introduced in chapter 3.2. The streams share parts of their trained weights θ and are defined as

$$\begin{aligned} h_{z_t}^{(m)} &\leftarrow \text{Attention}\left(Q = h_{z_t}^{(m-1)}, KV = h_{z \leq t}^{(m-1)}; \theta\right), \text{ (content stream: use both } z_t \text{ and } x_{z_t}) \\ g_{z_t}^{(m)} &\leftarrow \text{Attention}\left(Q = g_{z_t}^{(m-1)}, KV = h_{z<t}^{(m-1)}; \theta\right), \text{ (query stream: use } z_t \text{ but cannot see } x_{z_t}) \end{aligned} \tag{3.20}$$

The content stream $h_{zt}^{(m)}$ is initialized with the word embedding, while the query stream $g_{z_t}^{(m)}$ uses a random vector as starting point. It might seem unnecessary to have a representation that also includes the current token, since the model should never see the actual content of the position it is supposed to predict. But all kinds of contexts need to be available, since the model never knows in advance which factorization order it has to use. During finetuning, the content representation becomes the normal self attention mechanism, while the query representation is simply dropped.

In addition to using a permuted factorization order and a two stream attention mechanism, XLNet incorporates ideas from Transformer-XL (Dai et al. 2019). The first one is to use positional encoding that is based on the original sequence and not on the factorized permutation. In contrast to BERT, this positional encoding is not implemented in the embedding layer and added to the word embedding but is used in the attention layers (Dai et al. 2019, p.5). The relative distance between two positions is injected into the attention score. Furthermore, XLNet checks whether two positions are from the same segment, so this concerns the segment embedding that BERT uses. As a main benefit, XLNet can also process multiple segments as opposed to BERT.
The other Transformer-XL-based building block is the segment recurrence mechanism which enables the model to reuse the hidden states from previous segments. Segment A is defined as the sequence $\tilde{\mathbf{x}} = \left(x_1, ..., x_T\right)$ and segment B as $\mathbf{x} = \left(x_{T+1}, ..., x_{2T}\right)$ with permuted factorization orders $\tilde{z}$ and z of [1,...,T] and [T+1,...,2T] respectively. First, segment A is processed based on its respective factorization order. For every layer m the content representation $\tilde{h}^{(m)}$ is memorized. For the next segment, segment B, the attention update will be processed by additionally using the memories $\tilde{h}^{(m-1)}$. The memory is the content representation of the first segment of the previous layer. Together with the content representation of the second segment, this yields the input for the key and value for the content attention mechanism. The attention updates thus becomes:

$$g_{z_t}^{(m)} \leftarrow \text{Attention}\left(\text{Q} = g_{z_t}^{(m-1)}, \text{KV} = \left[\tilde{h}^{(m-1)}, h_{z<t}^{(m-1)}\right]\right) \tag{3.21}$$

$$h_{z_t}^{(m)} \leftarrow \text{Attention}\left(\text{Q} = h_{z_t}^{(m-1)}, \text{KV} = \left[\tilde{h}^{(m-1)}, h_{z \leq t}^{(m-1)}\right]\right) \tag{3.22}$$

3.7.1 Pretraining

In addition to the already mentioned BooksCorpus and English Wikipedia, the datasets Giga5, ClueWeb 2012-B and CommonCrawl are used but only for the large version of XLNet. XLNet base was trained on the BooksCorpus and English Wikipedia only which makes it more comparable to BERT. In addition to the usual sequence length of 512, XLNet uses a memory length that is 384. The large version was trained on 512 TPU v3 chips for 500K steps. As optimizer Adam[21] was used with a linear learning rate decay and a batch size of 2,048. Pretraining XLNet large took around 2.5 days. Except for the smaller dataset, the authors don't explicitly mention any other settings for the pretraining of XLNet base. It is therefore assumed that the same hyperparameters were chosen. Since the dataset is smaller, pretraining XLNet base probably took less than 2.5 days. XLNet large does not consider a next sentence prediction objective. For XLNet base the authors don't make an explicit statement. Since using a permutation-based approach, the model becomes harder to optimize and converges slower. Z. Yang et al. have thus decided to only perform partial prediction during pretraining, which means that only the last tokens in a factorization order are actually predicted. For unselected tokens that are not predicted, the query stream is not calculated which saves speed and memory in addition.

XLNet	Layers	Hidden size	Embedding size	Attention heads
base	12	768	768	12
large	24	1024	1024	16

Table 3.6: Available XLNet versions

3.7.2 Tokenization and embedding

The input that XLNet expects takes the form SEGMENT A [SEP] SEGMENT B [SEP] [CLS]. Since the positions are factorized and considered in the attention layers, they are not additionally embedded in the lowest embedding layer. Tokenization is done using SentencePiece which is explained in more detail in chapter 3.6.2.

[21] Go to appendix A.1 for more details on the Adam algorithm.

4

Fake News Challenge (FNC-1)

In 2017, the Fake News Challenge Stage 1 (FNC-1) was published. Organizers from the industry and academia created an online challenge accessible via `http://www.fakenewschallenge.org/` (Pomerleau and Rao 2017).

4.1 Background of the Fake News Challenge

The FNC-1 is conceptualized as an important pre-step in identifying Fake News and exploring how artificial intelligence tools can be leveraged in combatting them. The organizers argue that journalists are mainly in need of a semi-automated tool that can help them identify potential sources of Fake News rather than having a fully-automated pipeline that already detects the veracity of a news article itself. Furthermore, it is very challenging to create such a fully automated pipeline, since detecting Fake News requires fact-checking which can lead to elaborate investigations depending on the topic and the claim. Since manual fact-checking will still be necessary in the near future the FNC-1 focuses on identifying the relation between a short claim and different article bodies reporting about this claim. So given a certain claim about a topic, what are different news agencies reporting about this claim? If most news agencies agree with a claim, this can be interpreted as an indicator of the truthfulness of the claim. On the contrary, if a lot of news disagree with the claim, the claim is likely Fake News. Statistically speaking this idea is translated into a stance detection task with the claim being treated as a headline and the stance of the article body being either *Agree*, *Disagree*, *Discuss* or *Unrelated*. The first three labels are furthermore categorized as *Related*. Ultimately, Fake News Detection is thus treated as a classification task with four categories which are interpreted as the stance of an article body towards a given headline claim.

Registration for the FNC-1 opened in December 2016 and was closed in May 2017. In February 2017, the organizers released the training set[1] and published a baseline model a month later. In June, the test set was released with the final announcement of the results and winners taking place on June 15th 2017. According to Hanselowski et al., 50 teams participated in the challenge. The top 3 winners were chosen according to a specifically introduced evaluation metric and received cash prices of USD 1000, USD 600, and USD 400. The evaluation metric will be discussed in more detail in chapter 5.5.

[1]The training set is introduced and discussed with more details in chapter 5.1.

4.2 Baseline and participator's models

As already mentioned, the organizers released a baseline model along with the FNC-1 dataset which is introduced in chapter 5.1. The baseline model consists of a Gradient Boosting (Hastie, Tibshirani, and Friedman 2009, p.359ff.) classifier for which the *sklearn.ensemble* (Pedregosa et al. 2011) implementation is used. Gradient Boosting is based on an ensemble of decision trees, which are typically very unstable. The model is fed an array of features that are extracted from the training corpus. It uses a word overlap feature that compares the similarity of the words used in the headline to those used in the article body. Furthermore the identification of polarity and refuting words such as *fake*, *fraud*, *hoax* and *doubt* are extracted as a feature. For the overlap features the co-occurrence of uni-, bi- and 4-gram words and bi-, 4-, 8-, 16-gram characters in the headline and article body is used. In addition, the co-occurrence of stop words in the headline with the first 100 and 255 characters of the article as well as the article as a whole is used as a feature.
Most participating teams have used neural networks either as sole model architecture or as part of an ensemble model. The best three teams have all used some sort of neural network, out of which two used rather simple network architectures.

The winning team, SOLAT in the SWEN, used a weighted average of XGBoost decision trees (Chen and Guestrin 2016) and a deep convolutional neural network. The code is available under Pan, Sibley, and Baird 2017. The neural network consists of an embedding part, where headline and article body are embedded separately. It is followed by a stack of five convolution layers which are ultimately combined for headline and body and fed through three fully connected layers. The features that are fed to the boosting algorithm consist of TF-IDF[2], SVD[3], word2vec (Mikolov et al. 2013) and sentiment features.

The Athene (Ubiquitous Knowledge Processing Lab) team scored second on the leaderboard. The code is available under Prof. Dr. Gurevych 2017. They simply used a multi-layer perceptron with six hidden layers of varying dimensions with ReLU activation as defined in the appendix A.5. The last layer is a softmax layer for classification. The input for the model consists of a concatenated feature vector of headline, headline + body and body. Each feature vector consists of uni- and bi-grams of a 5,000 token vocabulary for words as well as characters and a topic modeling feature. According to the FNC-1 score, this team scored second best, but when using a macro-averaged F1 score (F_1-m) over the four classes, it is the winning team. The problem of choosing a useful metric for the given dataset is discussed in more detail in chapter 5.5 where the definition of F_1-m is given in equation 5.1.

The third prize went to the UCL Machine Reading team (Benjamin Riedel et al. 2017a). The team also made their code available via Benjamin Riedel et al. 2017b. Like Athene, they used a multi-

[2]For more details on the terms TF and TF-IDF, refer to appendix A.
[3]For more details on the role of SVD in NLP, refer to Jurafsky and Martin 2019, p.120ff.

layer perceptron but with one hidden layer only. The network also uses a ReLU activation[4] and a final softmax layer for classification. Furthermore, lexical and similarity features as well as bag-of-word representations[5] of the text input were used. For both headline and body the TF vector as well as the TF-IDF[6] was calculated separately. The TF-IDF vectors were used for a cosine similarity[7] between headline and body. The final concatenation of the TF, TF-IDF and the cosine similarity vectors were then fed into the network classifier.

[4] As defined in equation A.5 in the appendix A.
[5] As defined in Jurafsky and Martin 2019, p.58.
[6] For more details on the terms TF and TF-IDF, refer to appendix A.
[7] As defined in equation 3.13 in chapter 3.5.

5

Datasets and Data Pre-Processing

5.1 FNC-1

The basis for the FNC-1 dataset comes from the Emergent dataset (Ferreira and Vlachos 2016) which was created for an online journalism project about rumour debunking. The project is still running and a website with manually checked claims is available (Silverman 2019). Rumours about different topics including U.S. and world news as well as technology stories were extracted from websites such as *snopes.com* and twitter accounts such as *@Hoaxalizer*. From these various sources journalists first identified the respective claim and then searched for articles mentioning this claim. As a next step the journalists labeled the article as *For*, *Against* or *Observing* and then summarized the article into a headline. As an additional step the veracity level of the claim was labeled as *True*, *False* or *Unverified*. In total, 300 rumoured claims and 2,595 associated news with an average ration of 8.65 (7.31) articles per claim were considered. The dataset thus contains real world data which was manually labeled by journalists with regard to their stance and veracity level.

Claim: Robert Plant ripped up an $800 million contract offer to reunite Led Zeppelin
Source: mirror.co.uk (shares: 39,140) **Headline**: Led Zeppelin's Robert Plant turns down £500MILLION to reform supergroup **Stance**: *for*
Source: usnews.com (shares: 850) **Headline**: No, Robert Plant Didn't Rip Up an $800 Million Contract **Stance**: *against*
Source: forbes.com (shares: 3,360) **Headline**: Robert Plant Reportedly Tears Up $800 Million Led Zeppelin Reunion Contract **Stance**: *observing*
Veracity: *False*

Fig. 5.1: Example of a data point in the Emergent dataset
Source: Ferreira and Vlachos 2016, p.2.

For the FNC-1, organizers matched every article body with their respective headline and additionally created the fourth class *Unrelated* by randomly matching headlines and article bodies that belonged

to different topics. This additional class seems especially useful when thinking of phenomenons like clickbait where article authors try to generate clicks by either suggesting attention-grabbing headlines that have little or sometimes nothing to do with the stated claim of the headline. It therefore seems reasonable to train a model that is able to detect *Related* versus *Unrelated* text documents since it would be desirable to scrape through websites to look for claims possibly spreading Fake News and then automatically separate clickbait articles from actual articles covering the claim. Furthermore, 266 instances were created in addition to prevent participation teams from deriving labels for the test set since the Emergent dataset is publically available.

Headline: Hundreds of Palestinians flee floods in Gaza as Israel opens dams	
Agree (AGR)	Hundreds of Palestinians were evacuated from their homes Sunday morning after Israeli authorities opened a number of dams near the border, flooding the Gaza Valley in the wake of a recent severe winter storm. [...]
Disagree (DSG)	Israel has rejected allegations by government officials in the Gaza strip that authorities were responsible for released storm waters flooding parts of the besieged area. "The claim is entirely false, and [...]" [...]
Discuss (DSC)	Palestinian officials say hundreds of Gazans were forced to evacuate after Israel opened the gates of several dams on the border with the Gaza Strip, and flooded at least 80 households. Israel has denied the claim as “entirely false”. [...]
Unrelated (UNR)	A Catholic priest from Massachusetts had been dead for 48 minutes before he was miraculously resuscitated. However, it is his description about God that is bound to spark a hot debate about the almighty. [...]

Table 5.1: Example of a headline with several associated article bodies in the FNC-1 dataset

The class distribution over the four classes in the FNC-1 dataset is heavily skewed towards the *Unrelated* class. The 49,972 instances each consist of a headline (id est the claim to be looked at), the respective article body and label. In total, there are 1,669 unique article bodies and 1,648 unique headlines. The 300 topics are divided into 200 topics for training and 100 topics for testing. Not every claim is associated with all four labels.

5.2 FNC-1 ARC

Hanselowski et al. provide an in-depth analysis of the FNC-1 with a reproduction of the code and results of the three winning teams, the baseline model as well as a newly introduced model. In addition the authors introduce an extended dataset (FNC-1 ARC) for the stance detection task. The ARC dataset consists of 188 manually selected debate topics of popular questions from the user debate section of the New York Times. For each of these debate topics those user posts were selected

that were highly ranked by other users. These highly ranked user posts were processed by producing two opposing claims for them. Afterwards, crowd workers decided on the stance of the user posts with regard to the two opposing claims and labeled the post as either *Agree*, *Disagree* or *Discuss*. The *Unrelated* label was created by randomly matching user posts to different topics. The authors specifically mention topics like immigration, schooling issues and international affairs. As this dataset is built on user posts as opposed to the online news articles of the original FNC-1 dataset, it consists of shorter documents that tend to express one viewpoint only and are less balanced in their opinion as news articles. Using this additional dataset, the robustness of the provided models can thus be tested.

Example from the original ARC dataset		
Topic	Do same-sex colleges play an important role in education, or are they outdated?	
User post	Only 40 women's colleges are left in the U.S. And, while there are a variety of opinions on their value, to the women who have attended ... them, they have been ... tremendously valuable. ...	
Claims	**1.** Same-sex colleges are outdated **2.** Same-sex colleges are still relevant	
Label	Same-sex colleges are still relevant	
Generated instance in alignment with the FNC problem setting		
Stance	**Headline**	**Document**
AGR	Same-sex colleges are still relevant	Only 40 women's colleges are left in the U.S. ...

Fig. 5.2: Example of a data point in the original ARC dataset
Source: Hanselowski et al. 2018, p.9.

The extended FNC-1 ARC dataset combines the FNC-1 with the ARC datset. It consists of 64,205 instances are 14,233 instances more than the FNC-1 dataset. The label distribution is overall similar to the original dataset but the *Disagree* category has a bigger proportion. Different teams have reported this category to be the hardest to predict correctly, since it is so sparse compared to the other categories. It will be of special interest to see the change in performance of predicting this category when training the models on the FNC-1 versus the FNC-1 ARC dataset in chapter 6.4.

Dataset	Instances	Headlines	Article bodies	AGR	DSG	DSC	UNR
FNC-1	49,972	1,648	1,669	7.4%	1.7%	17.8 %	73.1%
FNC-1 ARC	64,205	1,834	6,023	7.7%	3.5%	15.3%	73.5%

Table 5.2: General information and distribution of class labels of training datasets

5.3 Descriptives

The raw datasets of the FNC-1 as well as the extended dataset FNC-1 ARC consist of the already described article bodies and respective headlines with stances. The average word length of the

headlines is below $T = 512$ which is the maximal token length for all models. This is before any data processing steps, such as the removal of stop words as well as the tokenization that is done differently with respect to the five models which is described in chapter 3.

Dataset	Headline	Body	Headline and body
FNC-1	11.13	369.70	380.83
FNC-1 ARC	9.98	309.89	319.88

Table 5.3: Average sequence length of words

5.4 Data Pre-Processing

Before evaluating the general performance of the models as well as analyzing a grid of hyperparameters, the raw data has to be processed. The data pre-processing steps were kept as minimal as possible.

5.4.1 Concatenation

As a first step, headlines and article bodies were concatenated into one long sequence with headline coming first and the respective article body following. While all models are capable of handling inputs that consist of two segments, it is interesting to analyze all models, when simply being fed a concatenated vector. In doing so, the models can be evaluated with respect to their capabilities in learning the semantical structures of one instance as a whole. Furthermore, the main difference between feeding the model with a two-segment input would be the additional [SEP] token between the headline and the article body. While it is true that the models can handle an input of two different segments the combined sequence length of both segments still cannot exceed $T = 512$ tokens. Using an additional special token might result in information loss for those instances that have a sequence length $T > 512$ and have to be truncated. The role of the sequence length will be examined in more detail in chapter 6.4 where sequence lengths of $T_1 = 256$ as well as $T_2 = 512$ are used for a grid search.
A different approach could be taken by breaking down the news articles into individual sentences and then concatenating every sentence with the respective headline. As discussed in chapter 3.4, Yinhan Liu et al. observed a loss in performance when doing so. The information of the individual article sentences belonging to the same document can't be fed to the model and is thus not considered by it. The model is then unable to learn long-range dependencies and the linguistic structure of a news article as a whole. This might be important in order to detect the stance correctly. Therefore, this approach is not further considered.

5.4.2 Tokenization

All model architectures can be used with their own respective tokenizers that were described in chapters 3.3.2, 3.4.2, 3.5.1, 3.6.2 and 3.7.2. In general, tokenizer algorithms use some sort of trade-off to deal with having a very flexible character-based versus a more efficient word-based tokenization. Since every tokenizer provides the possibility to process cased text, it was decided to not perform lower-casing.
The tokenizers were fed with text that was first processed with the *NLTK* word tokenizer which was necessary to remove stop words as will be discussed in the following chapter 5.4.3.

5.4.3 Stop Words and control characters

All tokenizers can detect and ignore control characters such as *"\r\n"*, *"\n\n"* and the indication of the suffix *n't* to auxiliary verbs via *"n\'t"*. It was thus not necessary to explicitly remove them.

NLP pre-processing usually consists of the removal of so called stop words. A stop word can be any word that is assumed to not carry any information and being unusually frequent. Different popular NLP-processing packages such as *NLTK* or *scikit-learn* offer different lists of stop words to be removed from any examined corpus. The advantage of using one of these available lists lies in the comparability in data pre-processing amongst different NLP tasks and easy handling. The disadvantage, however, is that different contexts and different tasks might require different treatment of stop words. The use of stop words has been discussed for sentiment analysis, where negation stop words like *not* usually flip the truth value of a statement (Reitan et al. 2015). The *scikit-learn* stop word list for example, contains negation words such as *not*, *never* or *nor*. As these words might potentially flip the truth value of a statement, they might carry valuable information in the context of Fake News detection which means removing them doesn't seem useful. Furthermore, traditional stop word lists have been criticized for which words were chosen as stop words by giving the impression of arbitrarily choosing words like *computer* as a stop word (Nothman, Qin, and Yurchak 2018).
With the rise of context-driven language models like BERT, another problem emerges: stop word lists also contain prepositions like *to*, *by* or *from* that occur with a high frequency. With the new paradigm of attention and context-based models these words have gained tremendous importance. One example is the question answering task that is internally processed when using a search engine such as Google. Google boosted its performance by implementing BERT just recently, simply because BERT is able to understand deeply semantic relationships transported by exactly these seemingly uninformative words. This enables BERT to understand a query such as "2019 Brazil traveler to usa need a visa" in the intended sense, which is Brazilian people traveling to the US. Before BERT, *to* was ignored since it was considered a stop word. The query thus resulted in information for US travelers to Brazil (Powell 2019). Although the context of the FNC-1 is not a question answering task, this example still shows how powerful the relatively new BERT architecture really is in addressing supposedly trivial words and processing their words appropriately. It was thus decided to keep the removal of seemingly un-

informative words as low as possible. The manually selected list contains the words *The*, *the*, *A*, *a*, *An*, *an*. As can be seen in table 5.3, the average sequence length of the raw datasets are lower than the maximum sequence length that the models allow for which is $T = 512$. The statistics given in this table concern the text before any type of tokenization or other data pre-processing, id est they are on a word and not on a token basis. In order to remove stop words, it was necessary to first tokenize the instances to be able to filter out the mentioned stop words. This leads to an increased average sequence length on a word basis, which can be explained by looking at the *NLTK* word tokenizer that was used in more detail. Using this tokenizer means that words such as *Nicaragua's* get split into two words, one for *Nicaragua* and one for *'s*. The concatenated average sequence length thus increases to 405.43 (from 369.7 before word tokenization) words for the FNC-1 and 340.37 words (from 309.89 before word tokenization) for the FNC-1 ARC dataset. As already mentioned in chapter 5.4.1, the sequence length is subject to more detailed evaluation that is conducted in chapter 6.4.
The organizers of the FNC-1 have implemented a traditional approach with their baseline by first tokenizing their documents using the *NLTK* word tokenizer and then removing all stop words from the *scikit-learn* list.

5.4.4 Padding and truncation

Traditionally, language models are restricted in their input lengths. For BERT-based models and XLNet this restriction lies with $T = 512$ input tokens per instance. All models use special tokens such as [CLS] and [SEP]. While the classification token [CLS] is used to predict the probability distribution over the four labels, [SEP] is used to indicate the ending of a segment. Since the input was fed as a single concatenated segment, only one [SEP] token was added per instance. After the tokenizers add these special tokens to each instance, the remaining tokens are padded using the [PAD] token of the respective model. For all BERT-based models this results in padding from the right while for XLNet the padding is done from the left. This can be explained by the input that the two model types expects. BERT processes an input of the form [CLS] TOKEN SEQUENCE [SEP], while the input form for XLNet takes TOKEN SEQUENCE [SEP] [CLS]. Instances that have a longer sequence length than 512 are truncated. As a final result, all tokens have a sequence length of $T = 512$ including all necessary special tokens. In addition, the model is fed with an identifier of which token is padded and which is not. This is necessary to only place the attention over the span of "true" tokens that is of non-padded tokens.

5.4.5 Data split

In order to properly evaluate the performance of a trained model, data should be split into a training, evaluation and test set (Hastie, Tibshirani, and Friedman 2009, p.222). The training and evaluation set are obtained by splitting the training data with a ratio of 80 to 20. Along with the baseline implementation, the organizers published a function to split the training data according to this ratio. This function was re-used resulting in the same training and evaluation set that was used by the baseline implemen-

tation as well as the other participating teams of the challenge for the FNC-1 dataset. The test set was made available by the challenge itself in the form of a separate file. In chapter 6, the basic performance of all five models is examined using the training and evaluation set. For the final grid search of chapter 6.4 that was conducted for each model separately, the best hyperparameter configuration with respect to the F_1-m metric was chosen on basis of the evaluation set. Each winning configuration was then tested by using the unseen data of the test set.

5.5 Evaluation metric

The performance of the participator's models of the FNC-1 was evaluated using a special metric. The main reasoning of using this metric was to account for the heavily skewed distribution within the class labels of the stances. As can be seen in table 5.2, almost three quarters of the instances in the FNC-1 dataset are classified as *Unrelated*.

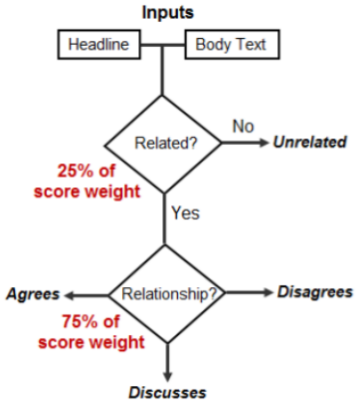

Fig. 5.3: Evaluation metric provided by the FNC-1 organizers
Source: Webpage of Pomerleau and Rao 2017.

An evaluation metric that would simply reward true positive predictions on the main diagonal of a confusion matrix could easily prefer models that always predict *Unrelated* since this prediction would be correct in most cases. Therefore, the organizers have decided to put a larger weight on the correct prediction of the *Related* class. Predicting *Unrelated* correctly is therefore less rewarded than predicting correctly that the stance belongs to any one of the three *Related* classes. Hanselowski et al. point out that this metric misses out on accurately considering the unequal distribution *within* the *Related* class labels. When only examining the three *Related* classes, 66.53% of the training instances are classified as *Discuss*, 27.7% as *Agree* but only 6.00% as *Disagree* label. While this may reflect the reality of many news articles that bring up various sources and arguments of a claim, it is of vital importance to accurately predict the *Agree* and *Disagree* classes since these classes facilitate the manual job of fact-checking a news article. If this uneven distribution is not considered in the evaluation metric, the model could just always predict *Discuss* as soon as the *Related* category is identified. The extended dataset FNC-1 ARC was also introduced in light of this unequal distribution and has around twice as

many instances that are classified as *Disagree*. The relative distribution within the *Related* categories for the FNC-1 ARC dataset is less therefore skewed. Around 57.76% of the training instances are classified as *Discuss*, 13.20% as *Disagree* and 29.05% as *Agree*.
Hanselowski et al. therefore propose to consider a macro-averaged F_1 score (F_1-m) as evaluation metric. It is defined as

$$F_1\text{-m} = \frac{1}{K}\sum_{k=1}^{K} 2 \cdot \frac{PR \cdot RE}{PR + RE} = \frac{1}{K}\sum_{k=1}^{K} \frac{2\#TP}{2\#TP + \#FP + \#FN} \tag{5.1}$$

with K being the number of classes in a multi-class setting, in this case $K = 4$. The true positive (TP), false positive (FP) and false negative (FN) cases are calculated for each class individually. Precision (PR) is defined as $\text{PR} = \frac{\#TP}{\#TP+\#FP}$ and recall (RE) as $\text{RE} = \frac{\#TP}{\#TP+\#FN}$. Precision measures the proportion of instances that were correctly classified as class k in relation to all instances that were classified as class k. Recall considers the correctly classified instances of k in relation to those instances that were wrongly classified as not k as well as the correctly classified cases. If a classifier would predict all instances as class k, one could achieve a perfect recall but a low precision (Goodfellow, Bengio, and Courville 2016, p.418). The F_1 metric considers this trade-off.
Another popular metric is the accuracy, defined as

$$ACC = \frac{\#TP + \#TN}{\#TP + \#TN + \#FP + \#FN} \tag{5.2}$$

with TN denoting the true negative cases and the other terms being the same as for the F_1-m metric. The accuracy is misleading in the case of an unbalanced class distribution, as will be seen in chapter 6.3. For this thesis, the relative FNC score (FNC_{rel}) and the accuracy ACC are reported in the initial evaluation step of experimentation but in light of the advantages of the F_1-m score, the main parameter of evaluating is F_1-m.

6 Hyperparameter Tuning

The use of large pretrained models for different NLP tasks is relatively new compared to other neural network application areas such as computer vision. Specific knowledge on how to receive the best possible results when finetuning is therefore less available. It is one of the main goals of this thesis to gain an understanding as to how much hyperparameter tuning is necessary for using large pretrained NLP models, how well transfer learning works in the specific context of stance detection of Fake News and how sensitive the models are to changes from default values for hyperparameters proposed by the authors. In this chapter, BERT, RoBERTa, DistilBERT, ALBERT and XLNet are evaluated with respect to the raised questions in two steps. As a first step, all five models are trained and evaluated on the datasets FNC-1 and FNC-1 ARC using the data split that was described in chapter 5.4.5. In doing so, the question of how the models perform in general can be answered. This is used as an indicator to determine how much hyperparameter tuning is necessary. Based on the results of this exploration step, an extensive grid search is conducted by defining a search space over the batch size, the maximum sequence length, the learning rate as well as the learning rate schedule. Furthermore, the exploration step evaluates different approaches in freezing techniques for finetuning. This is done by trying out 3 different freezing approaches of which the best performing one is used for the successive grid search. Before doing so, the general setup and implementation details are given in chapter 6.1, while the choice of values for the main hyperparameters of interest for finetuning is reflected in chapter 6.2.

6.1 General setup

The pretrained models are implemented using Python (Rossum 1995) and the deep learning framework PyTorch (Paszke et al. 2019). At the core of the implementation stands the *huggingface* library (Wolf et al. 2019) that makes a large variety of the SOTA models in NLP available and ready to use. The code is based on the *run_glue.py* script made available by *huggingface* (HuggingFace 2020a). The implementation makes use of the pretrained models that can be loaded and finetuned according to the specified task. Architecture-wise, the base implementation is used. For BERT, RoBERTa, DistilBERT, ALBERT and XLNet, different pretrained model versions are available. Table 6.1 gives the details of the used versions of *huggingface* for each model.

Model	Pretrained version	Task version
BERT	bert-base-cased	BertForSequenceClassification
RoBERTa	roberta-base	RobertaForSequenceClassification
DistilBERT	distilroberta-base	RobertaForSequenceClassification
ALBERT	albert-base-v1	AlbertForSequenceClassification
XLNet	xlnet-base-cased	XLNetForSequenceClassification

Table 6.1: Used implementation versions of each model from the *huggingface* pipeline.

The FNC-1 and FNC-1 ARC data are not lowercased which is why models that were pretrained on cased text corpora are chosen. For all models the implementation of sequence classification is used. This means that in addition to the model architecture itself an additional softmax classification layer is added in order to conduct a classification task. DistilBERT and XLNet don't have a feed-forward layer after the main layer stack. The sequence classification task of *huggingface* therefore added one to the implementation that is used for finetuning which means the weights of these pooling layers are randomly initialized in contrast to three other models that use pretrained pooler layers. DistilBERT is available in two versions: one that is distilled from the BERT base model and a second one distilled from the RoBERTa base model. Since RoBERTa outperforms BERT on multiple NLP tasks, it is of more interest how a lighter RoBERTa rather than a lighter BERT model performs. It is especially interesting to examine how much worse DistilBERT performs in comparison to its teacher as well as how its performance can be assessed compared to BERT. ALBERT is also available in two different versions. While the first version implements the original ALBERT, the second one dispenses with dropout, but uses additional training data and trains longer during pretraining. The additional data is the same as for XLNet large and RoBERTa. For this thesis, the simpler base version of ALBERT is chosen to assess the performance of this new approach in contrast to the other models.

In order to analyze the exploration steps as well as the hperparameter tuning experiments, TensorBoard is used. Hyperparameter tuning is conducted with *tune* (Liaw et al. 2018), a python library allowing for the integrated use of PyTorch, TensorBoad and a variety of search algorithms. For this thesis, a grid search was chosen as search algorithm. This has the advantage to be able to discover patterns of interactions between the hyperparameters of interest, although it is a very expensive algorithm that only allows to examine few hyperparameters due to combinatorial explosion.

All models are finetuned using an Ubuntu 18.04.3 LTS OS image with a maximum of 2 Tesla V100-PCIE-16GB GPUs. The virtual machine was created and accessed via the Leibniz Supercomputing Centre (lrz) lrz 2020.

Both evaluation steps that is the exploration and the grid search are performed with a variety of shared hyperparameter settings for all five models. All models were finetuned using the AdamW algorithm as defined in the appendix A.1 in equation A.9. Since a weight decay of $w = 0$ is chosen, AdamW reduces to the original Adam algorithm as defined in algorithm 1. Additional hyperparameters for Adam are set to the default values $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\delta =$ 1e-8 since Kingma and Ba have evaluated them to be relatively robust. Goodfellow, Bengio, and Courville confirm this and assert that the most important hyperparameter for Adam is the learning rate which will be evaluated in more depth with a grid search in chapter 6.4 (Goodfellow, Bengio, and Courville 2016, p.302). As warmup ratio, a value of 0.06 is chosen. This means that the first 6% of the total optimization steps are used to linearly increase the learning rate until the specified value is reached. The main intuition behind this is to perform more careful updates at the beginning of training when the model has not yet learned much. After the warmup the further learning rate schedule is suspect to more extensive evaluation in chapter 6.4. In order to avoid gradient explosion, the norm of all gradients is clipped with a maximum value of 1[1]. Gradients are not accumulated. For all models, the pretrained weights of the specified *huggingface* version are loaded once. The *huggingface* implementation already contains finetuned model architectures with additional classification layers for BERT, RoBERTa and ALBERT and an additional pooler as well as classifcation layer for DistilBERT and XLNet. This means that all weights of the pretrained models are always the same, while the weights for the finetuning layers are randomly initialized each time the model weights are loaded. For this reason, loading the models once minimizes this additional source of noise. The exploration step and grid search steps thus use the same randomly initialized weights for the finetuning layers per model[2]. In order to conduct the stance classification for the given datasets, the classification configuration of *huggingface* was changed from the default value of $k = 2$ to $k = 4$ output classes.
All other hyperparameters are evaluated in the exploration step in chapter 6.3 and grid search in chapter 6.4. The exploration step focuses on the hyperparameter freezing technique while the grid search evaluates a search space over the batch size learning rate, learning rate schedule and sequence length.

6.2 Benchmarking in NLP

The performance of a neural network largely depends on architecture choices such as the depth (number of (hidden) layers) and width (dimension of the (hidden) layers). In addition, many hyperparameters affect the performance. Selecting appropriate hyperparameter values can be difficult and requires in-depth understanding of a learning algorithm as well as necessary runtime and memory resources (Goodfellow, Bengio, and Courville 2016, p.422ff.). In the given case of finetuning large pretrained models like BERT and XLNet, the choice of hyperparameter is even more unclear since

[1] More details on the importance of gradient clipping can be found at Goodfellow, Bengio, and Courville 2016, p.402ff.
[2] Please note that the term *finetuning layer* refers to the additional layers that are put on top of the main model architecture, while *pretraining layers* are those layers that are part of all models no matter which finetuning task is used. The terms do not indicate which layers were updated during the finetuning. This is defined by the freezing technique which is evaluated in chapter 6.3

they largely depend on the specified task and dataset and the models can be used for a big variety of NLP tasks. Many different approaches and experimental setups, such as D. Liu and Miller 2020, X. Liu and Wangperawong 2019, Yang Liu 2019, Y. Wang et al. 2020 or X. Peng et al. 2020, are available that make use of the finetuning approach with different hyperparameter choices. Furthermore, the authors of BERT, RoBERTa, DistilBERT, ALBERT and XLNet give recommendations for hyperparameter choices when finetuning themselves. In order to determine which hyperparameter set is best used as starting point for the exploration step, the finetuning recommendations of the authors are considered. The authors give their recommendations with respect to finetuning their models on popular NLP benchmark datasets and tasks. Therefore, the most important benchmarks are presented and compared to the stance detection task.

All discussed model architectures provide some sort of evaluation of their pretrained models by finetuning them on different popular benchmarks in NLP research. However, the papers differ in how detailed they discuss their results and more importantly they do not all use the same benchmarks. Z. Yang et al. for example, publish a detailed list of all hyperparameters used for pretraining as well as finetuning for XLNet, while the evaluation of DistilBERT is kept relatively short. Since finetuning involves a specific task and dataset, the recommended hyperparameters depend on the benchmark dataset that was used. There are three major benchmarks that are widely used for evaluating NLP tasks, namely GLUE, SQuAD and RACE. Except for BERT and DistilBERT that are both not evaluated on the RACE dataset, all models use all of these three benchmarks.

SQuAD (Stanford Question Answering Dataset) was first introduced in 2016 and tackles the reading comprehension task. It is based on Wikipedia articles and aims to understand the reasoning necessary to answer a question (Rajpurkar et al. 2016). It is availabe in two versions. SQuAD2.0 consists of 100K questions out of which half are unanswerable, while SQuAD1.1 only contains answerable questions. The ability to answer a given question is introduced by presenting text passages that either contain or do not contain the answer.

RACE (ReAding Comprehension Dataset From Examinations) was officially introduced in 2017 and also attends to the reading comprehension task. It contains 100K questions from English exams for middle and high school Chinese students between 12 and 18 and was generated by English instructors (Lai et al. 2017, p.1). It covers a variety of topics and has a larger proportion of questions and also a longer average sequence length in comparison to other benchmark datasets such as SQuAD.

GLUE (General Language Understanding Evaluation) is the only one of these three benchmark that does not evaluate the performance with respect to one specific task but can be described as a multi-task benchmark. As such, the idea of not only establishing transfer *learning*, but in a sense also something like transfer *evaluation* is promoted. GLUE is supposed to favor and encourage models

that share general linguistic knowledge across tasks (A. Wang et al. 2018, p.3). In total, it consists of 9 different datasets covering question answering, similarity and textual entailment tasks. For final evaluation it calculates an average score over all 9 data problems. Furthermore, the datasets vary with respect to the covered topics, data sizes and degrees of difficulty. The different evaluation datasets and tasks require a model that is able to process single-sentence inputs as well as pairs of inputs. In creating this multi-task benchmark, models are favored to learn semantical and lexical structures beyond detecting trivial correspondences between inputs and outputs (A. Wang et al. 2018, p.1).

The datasets that are examined for this thesis tackle a stance detection task. For a given headline, the model is supposed to identify the stance of a news article and classify it as either *Agree*, *Disagree*, *Discuss* or *Unrelated*. Only considering the news article, one might first think of a sentiment analysis task but this ignores the main background of identifying the relationship between an article and a headline. Modelling this relationship is crucial, since the organizers have a semi-automated pipeline in mind where journalists can retrieve different articles that cover a certain claim to then identify potential sources of Fake News. Another widely used task is question answering which is also the basis for the RACE and SQuAD. While question answering tasks consider the relationship between two textual inputs, they focus on information retrieval. This might be interesting for future approaches to Fake News detection but is not suitable for the given data problem. Thus, out of the three domains of question answering, similarity and textual entailment, the latter seems to be the most appropriate task. Textual entailment is defined as NLI in the case of GLUE. NLI tasks consist of a sentence pair as input where one sentence is referred to as the *premise* and the other one as *hypothesis*. The task is now to decide if the premise sentence entails or contradicts the hypothesis or whether it is neutral. This is already closely related to the detection of the stance of an article with respect to a given headline.

In GLUE, there are four NLI tasks, namely MNLI, QNLI, RTE and WNLI. MNLI (Multi-Genre Natural Language Inference) is based on the Multi-Genre Natural Language Corpus as well as the Stanford Natural Language Inference dataset and can be seen as a classical implementation of the NLI task. QNLI (Question Natural Language Inference) relies on the Standford Question Answering Dataset and contains question-paragraph pairs. For GLUE, each paragraph was broken down into sentences and affiliated with the corresponding question to receive the necessary sentence-pair input. The task is then to find out whether the given paragraph sentence contains the answer to the question. The RTE (Recognizing Textual Entailment) dataset uses a binary classification of *Entailment* versus *No entailment*. The latter category contains the *Neutral* and *Contradiction* cases. The WNLI (Winograd Natural Language Inference) task is a reading comprehension task. For a given sentence a pronoun is selected. The task is to then determine the referent of that pronoun from a list of possible choices. For GLUE, sentence pairs are created by putting together the original sentences with a corrupted version of each sentence. For the corrupted version, the selected pronoun was replaced by each possible referent. The model has to predict whether the corrupted version is entailed by the

original sentence.
Out of the four tasks, the MNLI task of GLUE seems most appropriate to be compared to the stance detection task. The RTE case is also similar to the given task of the thesis, while QNLI and WNLI take a slightly different approach.

In addition to the three discussed benchmarks, some of the models are evaluated on less popular benchmarks. BERT uses the SWAG (Situations With Adversarial Generations) (Zellers, Bisk, et al. 2018) dataset for evaluating commonsense inference. DistilBERT and XLNet both use the IMDB dataset for a text classification task[3]. XLNet additionally uses four more datasets for evaluating its performance on text classification[4] as well as the dataset ClueWeb09-B (Lemur Project 2020) for a performance evaluation on document ranking.

In the following, the recommendations given in the papers are used as a basis that correspond to an NLI task of GLUE, preferrably the MNLI and RTE. Where this is not possible, recommendations for the next closest dataset and task are considered.

For BERT, Devlin et al. use a batch size of 32, finetune for 3 epochs and choose the best performing learning rate from a possible choice of {2e-5, 3e-5, 4e-5, 5e-5} for all GLUE tasks. For BERT large, finetuning was observed to be unstable on small datasets (Devlin et al. 2018, p.6) but since the given dataset is neither small nor is BERT large used, this does not concern the given task. The authors don't specify further which learning rate was used for which specific finetuning task of GLUE.
The authors of ROBERTa report their hyperparameter choices in more detail. However, the choices are still reported with respect to all GLUE tasks and not broken down by single tasks. As learning rate, a value among {1e-5, 2e-5, 3e-5} and a batch size of either 16 or 32 was chosen. Weight decay was always set to 0.1 and the learning rate schedule was linear with a warmup ratio of 0.06. The tasks are finetuned for a maximum of 10 epochs.
Sanh et al. don't report any specifics of the hyperparameters for finetuning for DistilBERT.
For ALBERT, Lan et al. report the chosen hyperparameter settings for each GLUE task individually. For the MNLI task a batch size of 128, a learning rate of 3e-5 and 1,000 warmup steps for 10,000 training steps are chosen which yields a much higher warmup ratio than RoBERTa. However, these details concern the ALBERT xxlarge version. It is not clear what choices might be best for ALBERT base since it doesn't outperform BERT anyway.
Z. Yang et al. also report hyperparameter choices for the MNLI task specifically. The batch size, learning rate and number of training steps are the same as for ALBERT. The authors have used a maximum sequence length of 128 instead of 512 (Z. Yang et al. 2019, p.16). An overview of the most

[3]Neither of the papers specify which precise dataset was used. Since there are different versions of the dataset, no source is indicated.
[4]The four additional datasets are Yelp-2, Yelp-5, Amazon-2 and Amazon-5. Neither of these four datasets can be identified distinctly since Z. Yang et al. do not specify the source of these datasets.

important choices can be found in table 6.2.

Model	Benchmark	Batch size	Learning rate	Learning rate decay	Training steps/Epochs
BERT	GLUE	32	{2e-5, 3e-5, 4e-5, 5e-5}	n.a.	3 epochs
RoBERTa	GLUE	{16,32}	{1e-5, 2e-5, 3e-5}	linear	maximum 10 epochs
DistilBERT	n.a.	n.a.	n.a.	n.a.	n.a.
ALBERT	GLUE-MNLI	128	3e-5	n.a.	10K
XLNet	GLUE-MNLI	128	3e-5	linear	10K

Table 6.2: Hyperparameter recommendations for each model

The different approaches of the authors when it comes to choosing hyperparameters indicates that there is not one optimal solution for all datasets.

6.3 Initial experiments

The goal of the exploration step is to evaluate the general performance and gain insights as to how useful the given recommendations for hyperparameter choices are. Therefore, the main hyperparameters of interest, namely the sequence length, batch size, learning rate and learning rate schedule are kept fixed to determine how well the models perform with respect to different freezing techniques. These conclusions are then considered for the grid search.

The exploration step takes the full maximum sequence length that is available for each model which consists of 512 tokens. The common assumption for batch sizes is that a higher batch size yields a more accurate estimate of the gradients (Goodfellow, Bengio, and Courville 2016, p.276). For the given hardware resources this results in a batch size of 8. This is considerably lower than the recommendations of table 6.2 but the best possible value for the given sequence length and memory capacity of the hardware.
The learning rate is set to 3e-5, in accordance with the recommendations for the MNLI task for ALBERT and XLNet and the general recommendations for BERT and RoBERTa. If reported, most authors recommend a linear learning rate schedule. It is thus chosen for the experimental step, too. The number of epochs is kept rather low with 2 epochs only. Since it is not the goal of the experimental step to receive the best possible performance but rather to gain first insights into the general adaptability of transfer learning for the stance detection of Fake News.
In addition, the models are trained with three different freezing techniques. When loading a pretrained model, there are different options for finetuning. It is possible to finetune all layers that is to use the pretrained weights as starting point and then update all parameter weights during finetuning. Another approach would be to only train the additional task-specific layers that are placed on top of the general model structure. For the setup of the initial experiments in this thesis, three different versions are tested. For the first run *Freeze*, all layers except for the last projection as well as the final classification

layer are frozen. The second run *No Freeze* uses no freezing at all which means that all parameters are updated during finetuning while the last run *Freeze Embed* is done with frozen embedding layers . In doing so, the models can be evaluated with respect to the linguistic and semantical power.

6.3.1 Results

The results are given in the overview tables 6.3 and 6.4 for the FNC-1 and the FNC-1 ARC dataset respectively. Following the conclusions drawn in chapter 5.5, the results are evaluated with the F_1-m metric. The accuracy is reported to show its ineffectiveness in the case of unbalanced class distributions and will not be further examined. The FNC_{rel} score is also misleading, as already discussed.

The first key learning is the importance of not freezing too many layers. All models have problems to accurately learn when the main layers (id est the encoder layers for the BERT-based and the relative-attention with feed-forward layers for XLNet) are frozen. Most models still predict every instance as *Unrelated* after two epochs. ALBERT performs best in this setting. This is not surprising since all encoder layers in ALBERT share weights and the model thus learns less information in general. Freezing the encoder layers therefore leads to less information loss for ALBERT. Interestingly, XLNet also performs slightly better compared to BERT, RoBERTa and DistilBERT. For the FNC-1 ARC dataset, BERT performs marginally better than RoBERTa and DistilBERT since it predicts four observations as *Disagree* after the second epoch. Since all three freezing setups were only performed once and the difference is so small it should not be overinterpreted however since it might be an artifact.

For both *No Freeze* and *Freeze Embed* all models are able to accurately learn to classify the given datasets. Even though finetuning was only performed for 2 epochs, the F_1-m score is already considerably good with a maximum of F_1-m = 82.80 for RoBERTa for the FNC-1 and F_1-m = 80.89 for XLNet for the FNC-1 ARC dataset. Especially for the FNC-1 dataset, the performances of XLNet and RoBERTa are very close. Most models perform better when the embedding layers are frozen. Only DistilBERT performs better when finetuning all layers which might be explained by the fact that it already has considerably less layers and thus less parameters to model the data which means every layer is important. This makes sense since DistilBERT distills BERT, respectively RoBERTa already as much as possible. Further reducing layers in the context of freezing during embedding thus hurts performance. For XLNet, the performance is slightly better in the *No Freeze* setup for the FNC-1 ARC dataset. With a difference of 0.54 points this could also be an artifact.

The main takeaway is that only finetuning the task-specific layers is not sufficient. Between not freezing any layers and freezing the embedding related layers, the difference in performance with respect to the F_1-m metric is not too big. Since the results are based on one run per freezing technique only, the robustness of the results is not fully evident. Starting the runs from afresh indicated that the performances can differ within a reasonable range which in some cases leads to a stronger

performance of the *Freeze Embed* and in others to a stronger performance of the *No Freeze* technique. For the grid search that follows in chapter 6.4, all models are finetuned with frozen embedding layers since freezing layers bears the advantage of speeding up training.

FNC-1

BERT	Freezing technique		
Metric	Freeze	No Freeze	Freeze Embed
Accuracy	71.69	93.92	94.86
F_1-m	20.88	70.36	**75.26**
FNC_{rel}	68.83	95.79	96.16
RoBERTa			
Metric	Freeze	No Freeze	Freeze Embed
Accuracy	71.69	95.41	95.82
F_1-m	20.88	80.83	**82.80**
FNC_{rel}	68.83	96.73	97.05
DistilBERT			
Metric	Freeze	No Freeze	Freeze Embed
Accuracy	71.69	94.40	94.42
F_1-m	20.88	**77.65**	75.85
FNC_{rel}	68.83	96.27	96.24
ALBERT			
Metric	Freeze	No Freeze	Freeze Embed
Accuracy	78.15	92.39	92.61
F_1-m	34.89	69.77	**71.13**
FNC_{rel}	78.35	94.31	94.60
XLNet			
Metric	Freeze	No Freeze	Freeze Embed
Accuracy	71.17	95.08	95.22
F_1-m	27.80	81.07	**82.55**
FNC_{rel}	69.92	96.38	96.37

Table 6.3: Results for the exploration step of the FNC-1 dataset. All models were trained once with three different freezing techniques. For *Freeze* only the last projection and classification layers are updated. For *No Freeze* all layers are updated, while for *Freeze Embed* all embedding-specific layers are excluded from updating.

FNC-1 ARC

BERT	Freezing technique		
Metric	Freeze	No Freeze	Freeze Embed
Accuracy	72.42	91.51	92.11
F_1-m	21.08	74.91	**76.62**
FNC_{rel}	70.05	93.27	93.62
RoBERTa			
Metric	Freeze	No Freeze	Freeze Embed
Accuracy	72.40	92.65	92.88
F_1-m	21.00	78.61	**78.89**
FNC_{rel}	70.02	94.31	94.57
DistilBERT			
Metric	Freeze	No Freeze	Freeze Embed
Accuracy	72.39	92.62	92.42
F_1-m	21.00	**78.48**	77.29
FNC_{rel}	70.02	94.24	94.13
ALBERT			
Metric	Freeze	No Freeze	Freeze Embed
Accuracy	76.99	89.07	89.53
F_1-m	34.63	69.11	**69.82**
FNC_{rel}	77.21	91.24	91.35
XLNet			
Metric	Freeze	No Freeze	Freeze Embed
Accuracy	71.94	93.22	93.09
F_1-m	25.85	**80.89**	80.35
FNC_{rel}	70.37	94.60	94.64

Table 6.4: Results for the exploration step of the FNC-1 ARC dataset. All models were trained once with three different freezing techniques. For *Freeze* only the last projection and classification layers are updated. For *No Freeze* all layers are updated, while for *Freeze Embed* all embedding-specific layers are excluded from updating.

The training process of the *Freeze Embed* variant is now further examined by looking at the training and evaluation loss. The main reasoning behind this is to check if the models show a tendency to overfitting as defined by Hastie, Tibshirani, and Friedman 2009. Overfitting is the phenomenon where a model learns random features of the training data that do not correspond to those features that determine the

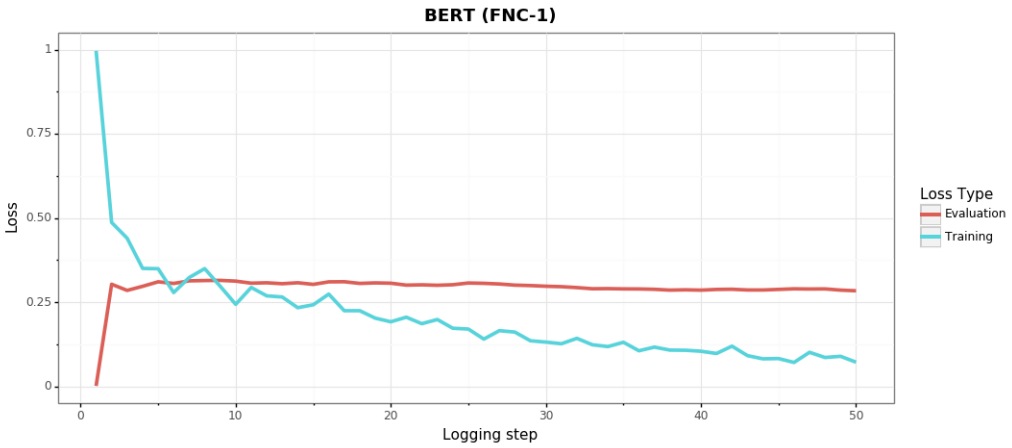

Fig. 6.1: Overfitting plot for BERT on the FNC-1 dataset. Every 50-th optimization step was tracked over the course of 2 epochs. There is no indication of overfitting as the evaluation loss stagnates.

actual underlying data-generating process. Thus, the gap between the training loss and the evaluation loss becomes too large. Typically the training loss decreases over the course of the optimization steps since this is precisely the idea of iteratively improving the weights of the parameters via a (stochastic) gradient descent-based algorithm. When overfitting occurs, the evaluation error starts to increase after a certain time. None of the five models show a strong indication of overfitting. All models have a decreasing training loss, as expected. The evaluation loss converges rather quickly and does not continue to increase. For all models, the evaluation loss is at a higher level for the extended FNC-1 ARC dataset, indicating that it is harder for the models to learn the the input structure adequately with a more heterogeneous dataset. Out of all five models, the learning for ALBERT appears to be the least stable, especially for the FNC-1 ARC dataset as can be seen in figure 6.2. In general, the models do not differ greatly in their development of the training and evaluation loss. As conclucion of this evaluation step, the number of epochs can be increased to further enhance the performances since since the evaluation metric F_1-m still increases and there is no indication of overfitting with respect to the evaluation dataset. For the grid search the number of epochs is therefore increased from 2 to 3.

6.4 Grid search

In machine learning, effective hyperparameter optimization has gained more and more attention. Usually, there is a tradeoff between using as little computational resources as possible and finding a model that yields the best possible performance, id est those with a minimal generalization error. When it comes to deep learning, a big amount of research is dedicated specifically to optimizing this tradeoff. Having varying amounts of data, algorithms that effectively scan through a pre-specified search space of hyperparameters are crucial. Traditionally a big focus is set on the finding of an

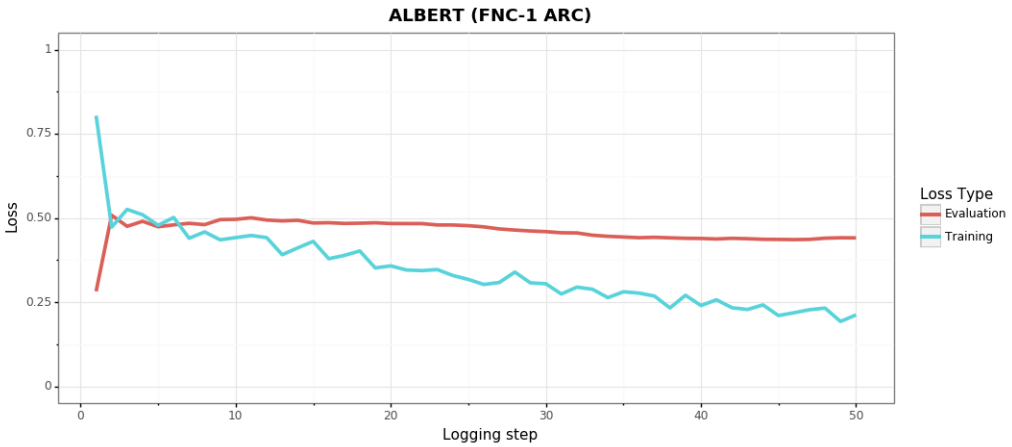

Fig. 6.2: Overfitting plot for ALBERT on the FNC-1 ARC dataset. Every 50-th optimization step was tracked over the course of 2 epochs. There is no indication of overfitting as the evaluation loss stagnates. In contrast to all other models, the training of ALBERT on the FNC-1 dataset converges slower and the evaluation loss is at a higher level.

optimal architecture such as, how many layers and neurons are to be used, which activation functions and which embedding technique work well. Furthermore there is usually a big importance planted on hyperparameters such as batch size, learning rate, momentum and weight decay.
In this setting, hyperparameter tuning is looked at from a different angle. Since traditional hyperparameter optimization focuses on training a model from scratch, some of the generated insights and algorithms might not be appropriate for the setting of transfer learning and finetuning (H. Li et al. 2020). There is little research on the effectiveness and necessity of hyperparameter optimization when finetuning, especially for NLP tasks. The biggest difference between pretraining and finetuning lies in the initialization of the weights. In the case of finetuning this initialization relies on the pretrained model which hopefully captures some intrinsic knowledge, whilst the pretraining phase works with random initialization or other parameter initialization frameworks. The goal of this experimental setup is thus to gain insights on exactly those two notions of effectiveness and necessity of hyperparameter tuning for finetuning in an NLP context. Therefore, a rather time-consuming but extensive approach is taken by choosing grid search for the hyperparameters batch size, maximal sequence length and type of learning rate schedule and learning rate.

The learning rate is probably the most common hyperparameter that is tuned. It is so important that Goodfellow, Bengio, and Courville state to only tune the learning rate, if one has only time to address one hyperparameter (Goodfellow, Bengio, and Courville 2016, p.417). For the finetuning setting, it is usually assumed that a drastically smaller learning rate should be used in comparison to pretraining (H. Li et al. 2020, p.1). During pretraining the model already learns a fairly good representation that is of value for any downstream task. The learned representation should therefore not be destroyed by

using a too big learning rate that strides away too fast from the already gained knowledge. In summary, the learning rate is so important because it regulates how much trust should be placed onto the current gradient while calculating the gradient of the next iteration. In order to cover a grid with different enough values and following the suggested grids for BERT and RoBERTa in table 6.2, the learning rates 1e-5, 2e-5, 3e-5 and 4e-5 are considered.

But not only the learning rate itself but furthermore the schedule that is used along with it plays an important role. In this context, three different schedule types are considered, namely a constant, linear and a cosine schedule. All three schedules use the warmup strategy for which a lower learning rate is used at the start of training to overcome optimization difficulties. The idea is to start with a smaller learning rate at the beginning of training, when the model has not yet learned much and thus should more careful in the updates of the gradients. The warmup ratio is 0.06 which means that the learning rate is linearly increased for the first 6% of the total optimization steps[5]. After the warmup period the targeted learning rate is reached. The schedules now differ in the successive handling of the learning rate. The constant schedule keeps the learning rate at the targeted value, while the linear and cosine schedule decay the learning rates accordingly. A visualization of the three schedules can be found in appendix A.2. It is expected that the linear and cosine schedule don't differ much in their performance but beat the constant schedule. Since ALBERT uses a different optimization algorithm during pre-training than the rest of the models it will be especially interesting to see if there is a difference in performance according to the learning rate schedules.

For the sequence length the naive assumption is that using as much context as possible is beneficial for the performance. A longer sequence length that doesn't truncate input sequences might therefore perform better. On the other hand it is imaginable that news articles might not need to be fed fully to the model since they might contain redundant discussion parts. Often, the main arguments are already shared at the beginning of the article with the full article further elaborating on the initially made statements. It might be sufficient to look at the beginning of articles which translates to a shorter sequence length. For the grid search, a sequence length of $T_1 = 256$ and $T_2 = 512$ are examined.

Lastly, a shorter sequence length bears the possibility to increase the batch size which was found to be rather low in the exploration step since the used virtual machine is limited in its memory capacities. A larger batch size is usually affiliated with a more accurate estimate of the gradient (Goodfellow, Bengio, and Courville 2016, p.276). On the contrary, smaller batch sizes often have a regularizing effect which might be due to the additional noise they add to the learning process. The values for the batch sizes are chosen such that a constant tokens per batch ratio is reached. Given the memory constraints, the highest possible batch sizes are 32 and 8 for a a sequence length of $T_1 = 256$ and $T_2 = 512$ respectively. An overview of the defined search space for the grid search is given in table 6.5.

[5]The total optimization steps are the number of training instances divided by the batch size and then multiplied by the number of epochs.

Hyperparameter		
Sequence length	256	512
Batch size	16, 32	4, 8
Learning rate	1e-05, 2e-05, 3e-05, 4e-05	
Learning rate schedule	constant, linear, cosine	

Table 6.5: Search space over chosen hyperparameters that is used for the grid search. The sequence length and batch size depend on one another due to memory capacity reasons of the virtual machine. For a sequence length of T_1=256, batch sizes of 16 and 32 and for T_2=512, 4 and 8 are considered. All three learning rate schedules use a warmup period of 6% of the total optimization steps.

6.4.1 Results

Given the defined search space in table 6.5, 48 combinations are evaluated for each model and dataset. The combinations are examined per learning rate and presented in table 6.6. Since the evaluation is split by looking at the four individual learning rates, for each model four winning configurations are reported. With five models being examined overall, this means that for each dataset 20 winning configurations are presented which yields an overview of 40 winning configurations in total.

When it comes to the sequence length, a value of 256 is preferred by most models for the FNC-1 dataset, while the preferred choice for the FNC-1 ARC dataset is 512, especially for RoBERTa and DistilBERT. This means that even though, the FNC-1 dataset contains longer sequences on average (see table 5.3), a shorter sequence length is preferred. This can be interpreted as an indication that the similarity of instances is more important than the average sequence length. Apart from the fact that the FNC-1 ARC dataset contains more instances compared to the FNC-1 dataset, the biggest distinction is that the additional instances were generated from a different context. Hanselowski et al. specifically introduced the extended dataset to test a proposed model's robustness by using the more heterogeneous FNC-1 ARC dataset. The general performance of the two different datasets is elaborated at a later point when discussing the indications of table 6.7.
For a sequence length of $T_2 = 512$ almost always a batch size of 8 is chosen. The only exceptions are found for a smaller learning rate of 1e-5. This is not surprising since a smaller batch size introduces more uncertainty in estimating the gradient. This uncertainty is then compensated by a smaller learning rate. For a sequence length of $T_1 = 256$ the affiliated batch size is more evenly distributed. For the FNC-1 dataset a sequence length of 256 is chosen 15 times in total with 7 configurations preferring a batch size of 16 and 8 configurations one of 32. For the FNC-1 ARC this even distribution can be reported as well, with a batch size of 16 being chosen 4 and a batch size of 32 5 times. Thus the choice of the batch size seems to be only important for a longer sequence length where a higher batch size is preferred unless the models are trained on a very small learning rate. For the FNC-1 ARC dataset a higher batch size of 32 tends to occur along with a higher learning rate of 4e-5.

		BERT		RoBERTa		DistilBERT		ALBERT		XLNet	
	LR	Winner	F_1-m	Winner	F_1-m	Winner	F_1-m	Winner	F_1-m	Winner	F_1-m
FNC-1	1e-5	16,256,cst	66.83	4,512,lin	**78.93**	32,256,cst	61.60	8,512,cst	58.56	16,256,cst	71.86
	2e-5	16,256,cos	**71.41**	32,256,cst	76.35	16,256,lin	69.98	8,512,cos	**63.55**	16,256,cos	70.90
	3e-5	32,256,lin	69.97	32,356,cst	75.63	32,256,cst	67.60	8,512,cos	61.78	8,512,lin	72.98
	4e-5	16,256,cos	71.05	16,256,cos	77.59	32,256,cos	**70.49**	32,256,cst	62.46	32,256,cst	**73.22**
FNC-1 ARC	1e-5	8,512,cst	68.10	4,512,cos	**76.88**	4,512,cst	71.83	4,512,cst	**64.71**	4,512,cos	**75.67**
	2e-5	16,256,cst	**71.04**	8,512,cos	76.75	8,512,lin	**73.57**	8,512,lin	64.26	16,256,lin	74.49
	3e-5	16,256,cst	70.53	8,512,lin	76.19	8,512,lin	71.44	32,256,cst	64.28	16,256,lin	75.05
	4e-5	32,256,lin	69.49	8,512,in	74.92	32,256,lin	72.49	32,256,lin	64.32	32,256,cos	74.61

Table 6.6: Overview over the results of the grid search with respect to the learning rate (LR) in the left. *Winner* denotes the winning configuration out of the 12 possible configurations per learning rate. The best configuration was always chosen with respect to the F_1-m metric on the evaluation set. The values in the *Winner* column indicate the batch size, the sequence length and the learning rate schedule in this order. The learning rate schedules are abbreviated, with *cst* denoting the constant, *lin* the linear and *cos* the cosine schedule. The winning configuration per model is indicated in bold. The two values in teal indicate the overall winning configuration per dataset, while the overall winning configuration over both datasets of RoBERTa with F_1-m = 78.93 is additionally marked by a box. All reported values are received on the final test set.

Evaluating the learning rate schedule, the most surprising finding is that the constant schedule is chosen most frequently with being the preferred choice in 15 of all 40 reported winning configurations. On a par with the constant schedule is the linear schedule that is preferred 14 times. It can be observed that most models choose the constant schedule (chosen 9 times) for the FNC-1 dataset compared to the linear schedule (chosen 4 times). For the FNC-1 ARC dataset the opposite is true with the linear schedule being preferred 10 times and the constant schedule 6 times. The cosine learning schedule is elected 11 times in total, with being the preferred choice 7 and 4 times for the FNC-1 and FNC-1 ARC dataset respectively. Overall the distribution is thus relatively even, with a slight tendence to either a constant or a linear schedule. Since the FNC-1 ARC dataset is more diverse in its instances a linear learning rate decay seems to be of more importance compared to keeping the learning rate constant after warm up. The warm up period seems to be more important than the decaying schedule that follows. There are no peculiarities when it comes to ALBERT which uses a different optimizer during pretraining.

When it comes to the learning rate, most models perform best for a smaller value of either 1e-5 or 2e-5. There is a difference in the two datasets however. For the FNC-1 ARC dataset the overall winning configuration is always either one of the two smaller learning rates for all models. Looking at the FNC-1 dataset, DistilBERT and XLNet yield an overall winning configuration with a learning of 4e-5 which is the highest considered learning rate of the grid. For DistilBERT this can be explained by the

fact that it only uses half the layers compared to its teacher RoBERTa which might require the model to stride away more from its pretrained version in comparison to the other models in order to adequately capture the given downstream task. For the FNC-1 ARC dataset this might not be observable since the heterogeneous dataset requires a smaller learning rate in general. Both DistilBERT and ALBERT perform relatively bad for a learning rate of 1e-5 on the FNC-1 dataset. Given the dataset of the finetuning task is relatively homogeneous in its instances, lighter BERT-based frameworks require a larger learning rate than 1e-5 in order to achieve a better performance. Further examining the FNC-1 dataset, it is interesting to see that XLNet is the only model that clearly seems to need a higher learning rate of either 3e-5 or 4e-5.

The general best performing learning rates of 1e-5 and 2e-5 are lower than the proposed learning rate of 3e-5 for the MNLI task for ALBERT and XLNet and is at the lower end of all suggested values of BERT and RoBERTa (see table 6.2). It seems that a more cautious updating of the pretrained parameters is important which is a strong indication of how well the large pretrained models already perform. H. Li et al. note that "it is believed that adhering to the original hyperparameters for fine-tuning with small learning rate prevents destroying the originally learned knowledge or features." (H. Li et al. 2020, p.1). A small learning rate is seemingly the most important factor for correctly using large pretrained NLP models on downstream tasks. BERT and RoBERTa prefer a learning rate of 2e-5 and 1e-5 respectively and show the same tendencies in performance with respect to the learning rate for both datasets. Thus this recommendation is relatively stable. DistilBERT and XLNet prefer a larger learning rate for the more homogeneous FNC-1 dataset and a smaller learning rate for the more heterogeneous FNC-1 ARC dataset. For ALBERT the learning rate should be larger than 1e-5 for the homogeneous dataset, while the impact of the learning rate is pretty small compared to the more heterogeneous FNC-1 ARC dataset. Later, it will become evident that ALBERT's much better performance for the FNC-1 ARC dataset can be explained largely by the improved prediction strength on the sparse category of *Disagree* instances. The more evenly distributed class labels of the FNC-1 ARC dataset have the biggest impact on ALBERT and XLNet.

Looking at the overall performance, ALBERT performs the worst and even worse than BERT. This confirms the statements of Lan et al. who have reported that ALBERT can only outperform BERT for the xxlarge and xlarge variant. Surprisingly, DistilBERT can outperform BERT on the FNC-1 ARC dataset. For the FNC-1 dataset BERT performs better however. When comparing the two different approaches of encoders versus autoregression, the best BERT-based model RoBERTa performs better than XLNet. For the FNC-1 dataset the performance is considerably better, while it is almost on par with XLNet for the FNC-1 ARC dataset. XLNet still outperforms BERT, DistilBERT and ALBERT on both datasets and is thus the second-best overall model. However, XLNet bears the additional disadvantage of training much longer than any BERT-based model. While RoBERTa trains for around 80 minutes for one configuration, XLNet takes around 180 minutes which corresponds to more than double the time of RoBERTa finetuning. The reason why XLNet does not beat RoBERTa might

lie in the specific advantage that XLNet introduces. Z. Yang et al. criticize BERT for making the assumption that all masked tokens in a sequence are independent. By using the PLM objective, XLNet can avoid making such an assumption is furthermore able to capture dependencies between tokens better than BERT. An example for this, was given in chapter 3.7. The given task of stance detection is not a token-level but a segment-level task. The big advantage of XLNet of better capturing the dependencies between tokens might thus be not of much use in the specific context of Fake News detection.
The overall winning model is RoBERTa with a batch size of 4, a sequence length of 512 and a learning rate of 1e-5. For the FNC-1 dataset, a linear schedule is preferred, while the finetuning on the FNC-1 ARC dataset renders a cosine schedule. As can be seen in figures A.2 and A.3 they don't differ as much which supports the general finding that the models are not very sensitive to different learning rate schedules.

The evaluation done so far focused on the F_1-m metric. Since both datasets consist of unevenly distributed and heavily skewed class labels, the class-wise F_1 metric is now considered to further understand the performances of the five different models. In general, the performance for DistilBERT, ALBERT and XLNet improves for the extended dataset FNC-1 ARC, while it slightly worsens for BERT and RoBERTa. All models except for RoBERTa can drastically improve their prediction strength for the *Disagree* class that has the fewest training instances, namely 1.7% and 3.5% for the FNC-1 and FNC-1 ARC dataset respectively. RoBERTa is the only model that performs worse on this category when being finetuned on a larger dataset. This could be interpreted as some sort of saturation effect. Using more evenly distributed data only helps to a certain degree. The FNC-1 ARC dataset is still heavily skewed and in addition more heterogeneous than the FNC-1 dataset. If the overall model architecture is already very powerful, as is the case for RoBERTa, the heterogeneity in the training instances might outweigh the positive factor of having more evenly distributed data.

Furthermore, XLNet was reported to perform almost on par with RoBERTa for the FNC-1 ARC dataset. This can be largely attributed to the boosted performance of the *Disagree* class. For the FNC-1 ARC dataset, XLNet reaches a value of F_1-DSG = 53.60 which is rather close to the RoBERTa values of 56.13 (FNC-1) and 55.11 (FNC-1 ARC). RoBERTa also performs very strongly because of its accurate predictions of the *Agree* class. For this class, XLNet performs surprisingly bad, especially on the FNC-1 dataset and it performs even worse than DistilBERT and about as good (or bad) as BERT. Only when using more data, XLNet is able to boost its performance compared to DistilBERT and BERT. Except for ALBERT, all models perform quite well on the *Discuss* class. All models, ALBERT included, perform extraordinarily good on the main class of the *Unrelated* instances. The minimum F_1-UNR value is 96.48 for ALBERT on the FNC-1 ARC and 96.64 on the FNC-1 dataset. There are no big differences in the very good performance for this class for the two datasets.
In total, RoBERTa is the strongest model. It also outperforms XLNet, for the FNC-1 remarkably and

for the FNC-1 ARC dataset only slightly. This better performance is not attributable to one specific class, since RoBERTa performs stronger on several classes. When considering F_1-m, DistilBERT, ALBERT and XLNet can improve their performance when finetuned on more data, while BERT and RoBERTa have a slight worse performance. Except for RoBERTa which is already quite strong, all models can improve their performance on the hardest to predict class *Disagree* when finetuned on more training instances.

In summary, the grid search showed the extraordinary strong performance of RoBERTa and it can outperform XLNet. This might be due to the specific Fake News detection task that is on a segment- rather than a token-level. In addition to performing better, RoBERTa also trains much faster than XLNet. The learning rate is the most important hyperparameter which corresponds to the current research. For BERT and RoBERTa a clear recommendation for the learning rate can be drawn, while DistilBERT, ALBERT and XLNet are more sensitive to different values of the learning rate. The sequence length is not as important and dominated by the similarity of training instances within a dataset. If the instances are more similar, the models manage finetuning well with shorter sequences. A more heterogeneous dataset requires the model to consider more context and thus a longer sequence length. The learning rate schedule is relatively unimportant.

		Dataset	
Model	Metric	FNC-1	FNC-1 ARC
BERT	**F_1-m**	**71.41**	**71.04**
	F_1-AGR	64.18	62.53
	F_1-DSG	41.11	45.02
	F_1-DSC	81.80	78.39
	F_1-UNR	98.55	98.24
RoBERTa	**F_1-m**	**78.93**	**76.88**
	F_1-AGR	73.53	70.17
	F_1-DSG	56.13	55.11
	F_1-DSC	86.82	83.16
	F_1-UNR	99.24	99.07
DistilBERT	**F_1-m**	**70.49**	**73.57**
	F_1-AGR	66.72	65.52
	F_1-DSG	32.83	47.60
	F_1-DSC	83.62	82.45
	F_1-UNR	98.80	98.71
ALBERT	**F_1-m**	**63.55**	**64.72**
	F_1-AGR	54.74	56.13
	F_1-DSG	27.37	33.25
	F_1-DSC	75.46	73.01
	F_1-UNR	96.64	96.48
XLNet	**F_1-m**	**73.22**	**75.67**
	F_1-AGR	64.98	68.04
	F_1-DSG	48.50	53.60
	F_1-DSC	81.18	82.20
	F_1-UNR	98.22	98.85

Table 6.7: Overview of all model's performances with respect to class-wise F_1 as well as F_1-m in comparison to the two datasets FNC-1 and FNC-1 ARC.

7 Conclusion and Outlook

The outline of this thesis was to evaluate unsupervised representation learning in the context of Fake News detection. In order to do so, Fake News was defined as a verifiably wrong text piece that is spread with a malicious intent. It was shown how humans are generally susceptible to Fake News due to confirmation bias, the echos chamber effect and general sociological needs. Information ecosystems were explained from a prospect theory point of view where publishers and consumers seek to maximize their utilities when making choices. Fake News can especially thrive in a context where publishers focus on the short-term interest of increasing their readership and the consumers' utility is dominated by psychological and sociological aspects rather than the need for unbiased information.
The Fake News Challenge Stage 1 provides a specific task and dataset to tackle the challenge of Fake News detection. The main reasoning behind the challenge is to create a semi-automated pipeline where the stance of different news articles towards a certain claim is examined. The dataset thus contains instances with a headline, article body and one of the four labels *Agree*, *Disagree*, *Discuss* and *Unrelated*. A second dataset was used that is enriched by using instances created from a different source and with a less skewed label distribution.
The resulting stance detection task was then evaluated by using five popular pretrained NLP models, namely BERT, RoBERTa, DistilBERT, ALBERT and XLNet. The era of largely successful transfer learning in NLP began with BERT in 2018. BERT was the first pretrained NLP model to successfully exploit the Transformer architecture and incorporate bidirectional context without recurrence. RoBERTa further enhanced BERT's performance by excessively pretraining the BERT structure on more data for a longer time. DistilBERT and ALBERT have raised questions of usability and focused on creating lighter frameworks with a worse performance for the sake of an improved usability. While DistilBERT uses a distinct knowledge distillation approach and drastically reduces the number of layers, ALBERT does not really form "A Lite BERT" as it claims. ALBERT can outperform BERT but only when increasing the width of its network which results in an upscaled hidden size dimension. XLNet introduces a new approach and positions itself in contrast to the BERT and BERT-based encoder model architecture. XLNet does so by reintroducing the idea of autoregression which was historically very popular in NLP and is better able to capture dependencies between tokens.
In order to accurately use the two datasets FNC-1 and FNC-1 ARC, the headlines and article bodies were concatenated and a manually defined list of stop words was removed. The evaluation of the five models was two-fold by conducting an initial experiment setup that was followed by an extensive grid search. The main conclusion drawn from the experimental step is the importance of not freezing

too many layers. While all five models have learned powerful feature representations, it does not suffice to only finetune the classification and last pooling layer for the stance detection datasets. It was further experimented to only freeze the embedding layers which resulted in an already convincing performance after two epochs only. Since there were no signs of overfitting, the number of epochs was thus increased to 3 for the next evaluation step.
The grid search was conducted over a search space including the batch size, sequence length, learning rate as well as the learning rate schedule which resulted in 48 combinations per model and dataset. The analysis was done with respect to the learning rate as indicated in table 6.6. The key conclusions are as follows. For the sequence length, most models preferred a value of 256 for the FNC-1 and a value of 512 for the FNC-1 ARC dataset. For a sequence length of 512, almost all models chose a higher batch size of 8, with the only exceptions occurring for a small learning of 1e-5. For a sequence length of 256, the batch size was evenly distributed over both datasets and also for each dataset separately. Thus, the for a sequence length of 256, batch size is no important hyperparameter. When examining the learning rate schedule the main conclusion is that it does not pose an important hyperparameter either. The constant schedule was chosen surprisingly often, especially for the FNC-1 dataset. For the FNC-1 ARC dataset, most models preferred a linear schedule. Most arguably the most important aspect is that all three schedulers use a warmup period. The models are thus found to be relatively robust with respect to the schedule choice.
The most decisive hyperparamter is the learning rate. In general, a smaller learning rate is better. For BERT and RoBERTa the learning rate pattern is relatively robust which makes it possible to recommend a learning of 2e-5 and 1e-5 respectively which are below the MNLI recommendations that ALBERT and XLNet and at the lower end of the recommendations for BERT and RoBERTa. DistilBERT and XLNet prefer a larger learning rate for the more homogeneous FNC-1 dataset and a smaller one for the more heterogeneous FNC-1 ARC dataset. For ALBERT, the learning rate does not have a distinct impact for the larger FNC-1 ARC dataset but is more important on the smaller and more skewed FNC-1 dataset. In that case, the learning rate for ALBERT should be above 1e-5.
RoBERTa is the overall winner, for the FNC-1 dataset it outperforms XLNet remarkably and for the FNC-1 ARC dataset just slightly. The autoregressive approach of XLNet is thus less promising, especially when taking the much longer training time into account. The main conclusion is therefore that the excessive pretraining of RoBERTa beats the smart idea of PLM that XLNet proposes. RoBERTa is in fact so powerful that even a distilled version from it, namely the used DistilBERT architecture, can outperform BERT on one of the datasets. The reason why XLNet does not beat RoBERTa might lie in the specific advantage that XLNet introduces. Z. Yang et al. criticize BERT for making the assumption that all masked tokens in a sequence are independent. By using the PLM objective, XLNet can avoid making such an assumption is furthermore able to capture dependencies between tokens better than BERT. An example for this, was given in chapter 3.7. The given task of stance detection is not a token-level but a segment-level task. The big advantage of XLNet of better capturing the dependencies between tokens might thus be not of much use in the specific context of

Fake News detection.
ALBERT base performs as bad as expected, since it relies on parameter sharing only the xlarge and xxlarge versions are reported to outperform BERT. All models but RoBERTa can drastically improve their prediction strength on the *Disagree* class. This indicates that using a more evenly distributed dataset for sure helps to improve the performance but only to a certain degree.

The overall conclusion that can be drawn from this thesis is how powerful and strong the unsupervised representation learnings of BERT, RoBERTa, DistilBERT, ALBERT and XLNet are. Even with minimal hyperparameter tuning and only finetuning for 2 epochs, the models already performed considerably good on both datasets. With respect to the considered hyperparameters the two most important conclusions are to not only finetune the classification and pooling layers that are put on top of the pretrained models[1]. Secondly, the most important hyperparameter is the learning rate. The recommendation of Goodfellow, Bengio, and Courville to focus on the learning rate, when one has only time to address one single hyperparameter can be confirmed. Furthermore, the recommendations that the authors give should be considered with care, especially when the task differs from the one that was used for the recommendation. Even though the NLI task was found to be close to the stance detection task, the grid search yielded different recommendations and results compared to those of the authors. At last, the models are relatively robust with respect to the learning rate schedule, the batch size, as long as it is adjusted to the learning rate and to a certain degree also the sequence length. Furthermore, the excessive pretraining approach of RoBERTa can outperform the PLM objective of XLNet, arguably for the case of the segment-level stance detection task.

In future research, it is interesting to systematically evaluate the importance of data pre-processing steps. One could further examine the necessity of stop words removal by not excluding any or by using one of the pre-defined lists that are widely used within the NLP community. Furthermore, for this thesis the headlines and article bodies were fed as one concatenated input to the models. It would be of interest to experiment with using the headlines and article bodies as two segments to analyze how this impacts the models' performances. RoBERTa reported a loss of performance during pretraining. It would therefore be especially valuable to study the impact on RoBERTa's performance (Yinhan Liu et al. 2019, p.5).
It was hypothesized that the robustness of the learning rate schedule is due to the fact that all three examined schedules use a warmup period. Thus, the role of the warmup period could be analyzed in more detail. A setup where no warmup period is used versus one with a prolonged period in comparison to the here used 6% could yield more insight into the function and importance of learning rate schedule for finetuning. Likewise, an experimental setup with different optimizer algorithms could enhance the understanding of finetuning procedures in NLP. This is also interesting since Lan et al. distinctively use a different optimizer to boost performance during pretraining. Another

[1]For some models the pooling layer is already incorporated in the main architecture but still it does not suffice to only finetuning the last pooling and classification layer.

hyperparamter of interest is the value of the weight decay which was set to 0 for finetuning in this thesis.
Another interesting approach would be to examine different finetuning layers and architectures. For this thesis, the finetuning architectures made available by *HuggingFace* were used. For a classification task, only the last hidden state of the [CLS] token is fed to the classification layer. In future, more understanding of the internal workings can be gained by using different aggregations of hidden states for the last layer. Likewise, the freezing techniques can be further analyzed by gradually unfreezing all layers and deriving further insights on how important the individual main layers are.

When it comes to Fake News detection in general, a promising research direction is the incorporation of knowledge bases. As already outlined, the main assumption of the FNC-1 is that a semi-automated pipeline is of most use for practicioners in the news industry. The way the challenge is constructed also means that manual fact-checking is still necessary. Meanwhile, there are approaches to use models like BERT as knowledge bases instead of traditional structured knowledge bases. Petroni et al. report that the pretrained BERT already contains relational knowledge which means there is a large potential of using BERT as unsupervised open-domain QA systems. Further elaborating on this, might enable a pipeline that also includes fact-checking by making queries to BERT.
Another approach is taken by Zellers, Holtzman, et al. who have introduced GROVER, a model that is not only able to detect but furthermore to actually *write* Fake News. This might seem counterintuitive in combatting Fake News at first. The authors argue however that in order to fully understand how to protect against a threat it is of great value to understand the process and emergence of this threat in the first place. This is possible when one is able to accurately model the threat.

In the end, the different directions, approaches and modelling techniques to fight Fake News will only be as successful as their adaption to real-word processes. This also contains the already mentioned ecosystem and utilities that not only publishers but also consumers gain by sharing and consuming Fake News. AI tools to detect Fake News can thus be only seen as one brick in mitigating the negative effects that Fake News actually entail.

A

Additional Formulas

The chain rule, also referred to as product rule, that is used for (permutation) language models is defined as follows. For a set of random variables $X_1, ...X_T$ it holds that

$$P\left(\bigcap_{k=1}^{T} X_k\right) = \prod_{k=1}^{T} P\left(X_k | \bigcap_{j=1}^{k-1} X_j\right) \tag{A.1}$$

TF-IDF (Term Frequency-Inverse Document Frequency) indicates the importance of a term t in a given document d in relation to other documents. The measure is calculated as the product of the TF (Term Frequency) and the IDF (Inverse Document Frequency). TF simply counts the occurence of term t in a given document d with T terms in total which will result in a higher value if a term occurs more times compared to a term that occurs less frequently. Terms like *the* or *a* often occur in a document but don't transport a lot of information. IDF therefore weighs down frequent terms and scales up less frequent terms. This is done by putting the total number of documents $\#d$ in relation to those document in which the term t appears ($\#d_t$). A value of 0 implies that a term is not very informative while higher values imply more information (Thanaki 2017, p.159ff.).

$$\text{TF-IDF}(t, d) = \text{TF} \cdot \text{IDF} = \frac{\#t}{\sum_{t=1}^{T} \#t} \cdot \log\left(\frac{\sum_{d=1}^{D} \#d}{\sum_{d_t=1}^{D_t} \#d_t}\right) \tag{A.2}$$

The softmax function is defined as follows (Goodfellow, Bengio, and Courville 2016, p.179):

$$softmax(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{A.3}$$

The Gaussian Error Linear Unit (GELU) function is defined as follows (Hendrycks and Gimpel 2016):

$$\begin{aligned} \text{GELU}(x) &= xP(X \leq x) = x\Phi(x) \\ &\approx 0.5x\left(1 + \tanh\left[\sqrt{2/\pi}\left(x + 0.044715x^3\right)\right]\right. \end{aligned} \tag{A.4}$$

The Rectified Linear Unit (ReLU) function is defined as follows (Agarap 2018):

$$\text{ReLU}(x) = \max(0, x) \tag{A.5}$$

A.1 Optimization algorithms

BERT, RoBERTa, DistilBERT and XLNet use the Adam optimizer (Kingma and Ba 2014) for pretraining. Adam (Adaptive Moments) is an optimization algorithm that relies on adaptive learning rates. The algorithm is defined as follows (Goodfellow, Bengio, and Courville 2016, p.301):

Require: Learning rate ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, β_1 and β_2 in $[0, 1)$.
(Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$

Initialize 1st and 2nd moment variables $\boldsymbol{s_0 = 0}$, $\boldsymbol{r_0 = 0}$
Initialize time step $t = 1$
while *stopping criterion not met* **do**
  Sample a batch of m examples from the training set $\left\{x^{(1)}, \ldots, x^{(m)}\right\}$
  with corresponding targets $\boldsymbol{y^{(i)}}$
  Compute gradient: $\boldsymbol{g_t} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), \boldsymbol{y}^{(i)}\right)$
  Update biased first moment estimate: $\boldsymbol{s_t} \leftarrow \beta_1 \boldsymbol{s_{t-1}} + \left(1 - \beta_1\right)\boldsymbol{g_t}$
  Update biased second moment estimate: $\boldsymbol{r_t} \leftarrow \beta_2 \boldsymbol{r_{t-1}} + \left(1 - \beta_2\right)\boldsymbol{g_t} \odot \boldsymbol{g_t}$
  Correct bias in first moment: $\hat{\boldsymbol{s}}_{\boldsymbol{t}} \leftarrow \frac{\boldsymbol{s_t}}{1-\beta_1^t}$
  Correct bias in second moment: $\hat{\boldsymbol{r}}_{\boldsymbol{t}} \leftarrow \frac{r_t}{1-\beta_2^t}$
  Compute update: $\boldsymbol{\Delta\theta_t} = -\epsilon\frac{\hat{\boldsymbol{s}}_t}{\sqrt{\hat{\boldsymbol{r}}_t}+\delta}$ (operations applied element-wise)
  Apply update: $\boldsymbol{\theta_{t+1}} \leftarrow \boldsymbol{\theta_t} + \boldsymbol{\Delta\theta_t}$
  $t \leftarrow t + 1$
end

Algorithm 1: The Adam algorithm

Adam uses bias-corrected estimates of the first and second moment and individually adapts the learning rate ϵ to all model parameters contained in $\boldsymbol{\theta}$. In doing so, the algorithm is able to have decreased learning rates for parameters with larger partial derivatives and vice versa. However, the Adam algorithm has been observed to not generalize as well, especially compared to Stochastic Gradient Descent with (Nesterov) Momentum (Goodfellow, Bengio, and Courville 2016, p.289ff.). It therefore has been extended to incorporate a weight decay which results in the AdamW algorithm (Loshchilov and Hutter 2018). For finetuning, *huggingface* uses the AdamW optimizer.

In a general gradient descent setting, the update at iteration $t+1$ for the weights $\boldsymbol{\theta}$ of a neural network is calculated by applying

$$\boldsymbol{\theta_{t+1}} \leftarrow \boldsymbol{\theta_t} - \epsilon \boldsymbol{g_t} \tag{A.6}$$

with $\boldsymbol{g_t}$ denoting the gradient of a loss function L with respect to $\boldsymbol{\theta}$ and ϵ the learning rate. When a weight decay is introduced, the scalar w poses an exponential decay where the gradients of the previous iteration is scaled down more and more with increasing training steps. The parameter update for $\boldsymbol{\theta}$ thus becomes

$$\boldsymbol{\theta_{t+1}} \leftarrow (1-w)\boldsymbol{\theta_t} - \epsilon \boldsymbol{g_t} \tag{A.7}$$

Since it would interfere with the calculation of the moving averages of the mean and the variance estimates, the weight decay is performed after the parameter-wise learning rate has been adjusted. The update term $\boldsymbol{\Delta\theta_t}$ is thus defined as

$$\boldsymbol{\Delta\theta_{t,AdamW}} = -\left(w\boldsymbol{\theta_t} + \epsilon \frac{\hat{\boldsymbol{s}}_t}{\sqrt{\hat{\boldsymbol{r}}_t} + \delta} \right) \tag{A.8}$$

which results in the overall update of

$$\begin{aligned} \boldsymbol{\theta_{t+1}} &\leftarrow \boldsymbol{\theta_t} + \boldsymbol{\Delta\theta_{t,AdamW}} \\ \Leftrightarrow \boldsymbol{\theta_{t+1}} &\leftarrow \boldsymbol{\theta_t} - w\boldsymbol{\theta_t} - \epsilon \frac{\hat{\boldsymbol{s}}_t}{\sqrt{\hat{\boldsymbol{r}}_t} + \delta} \\ \Leftrightarrow \boldsymbol{\theta_{t+1}} &\leftarrow (1-w)\boldsymbol{\theta_t} - \epsilon \frac{\hat{\boldsymbol{s}}_t}{\sqrt{\hat{\boldsymbol{r}}_t} + \delta} \end{aligned} \tag{A.9}$$

A.2 Learning rate schedules

Using a learning rate schedule η_t, the update rule for the parameters to be trained becomes

$$\theta_{t+1} \leftarrow \theta_t + \eta_t \epsilon g_t \tag{A.10}$$

The following learning rate schedules were used for the grid search presented in chapter 6.4. The figures show the warming up period of 100 training steps during which the learning rate is linearly increasing to its specified value. The schedules then differ in their decay structure. The constant schedule depicted in figure A.1 has no decay and simply keeps the same learning rate ϵ over all training steps after the warm up period. The linear schedule and cosine schedule inf figures A.2 and A.3 use a linear and cosine decay structure for ϵ respectively after the initial warming up period.

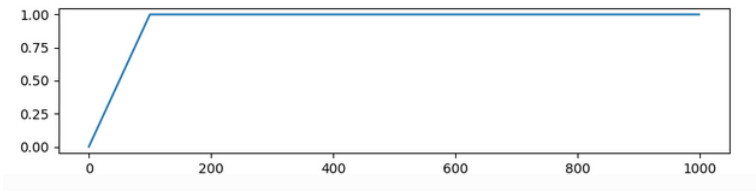

Fig. A.1: Constant learning rate schedule with no decay after 100 warmup steps.
Source: Webpage of HuggingFace 2020b.

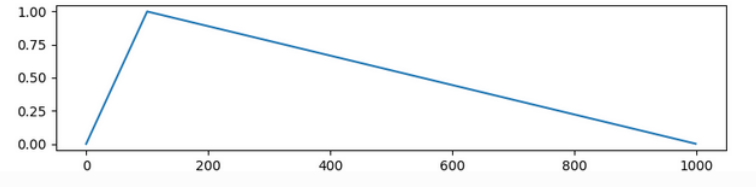

Fig. A.2: Linear learning rate schedule with linear decay after 100 warmup steps.
Source: Webpage of HuggingFace 2020b.

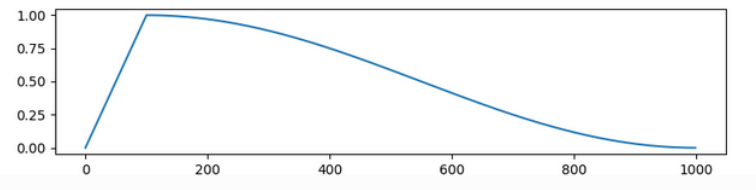

Fig. A.3: Cosine learning rate schedule with cosine decay after 100 warmup steps.
Source: Webpage of HuggingFace 2020b.

B

Language Models Details

In this chapter, a variety of additional information about the five discussed models are shared with the aim of enabling the reader to better compare them to another.

Model	Published By	Publication Date
BERT	Google AI Language	October 2018
RoBERTa	Facebook AI & University of Washington	July 2019
DistilBERT	HuggingFace	August 2019
ALBERT	Google Research & Toyota Technological Institute at Chicago	September 2019
XLNet	Carnegie Mellon University & Google Brain	June 2019

Table B.1: Publication details of all five models

All BERT-based models use Masked Language Modeling as a pretraining objective. XLNet relies on a Permutation Language Modeling task. BERT and ALBERT additionally consider segment coherence by implementing the Next Sentence Prediction task (BERT) and the Sentence Ordering Prediction (ALBERT). RoBERTa relies on the MLM task only, while DistilBERT implements the distillation objective that incorporated the similarity embedding and cross-entropy based loss to align the student to the teacher. An overview is given in table B.2.

Model	Pretraining objective	Batch size	Learning rate
BERT	MLM & NSP	256	1e-04
RoBERTa	MLM	8,000	1e-03
DistilBERT	MLM & Distil	up to 4,000	n.a.
ALBERT	MLM & SOP	4,096	0.00176
XLNet	PLM	2,048	1e-05

Table B.2: Details of pretraining parameters of every model.

Model	Embedding Model	Vocabulary size	Data
			BooksCorpus, English Wikipedia +
BERT	WordPiece	28,996	-
RoBERTa	Byte-level BPE	50,265	CC-News, OpenWebText, Stories
DistilBERT	Byte-level BPE	50,265	CC-News, OpenWebText, Stories
ALBERT	SentencePiece	30,000	-
XLNet	SentencePiece	32,000	Giga5, ClueWeb 2012-B, CommonCrawl

Table B.3: Details of data basis and text processing used for pretraining of every model. For BERT the vocabulary size of the cased version is indicated. For the uncased version, BERT has a vocabulary size of 30,522 tokens. For XLNet it is not clear if the additional data was only used for pretraining XLNet large or also XLNet base. BPE stands for byte-pair encoding.

C

Additional Evaluation Results

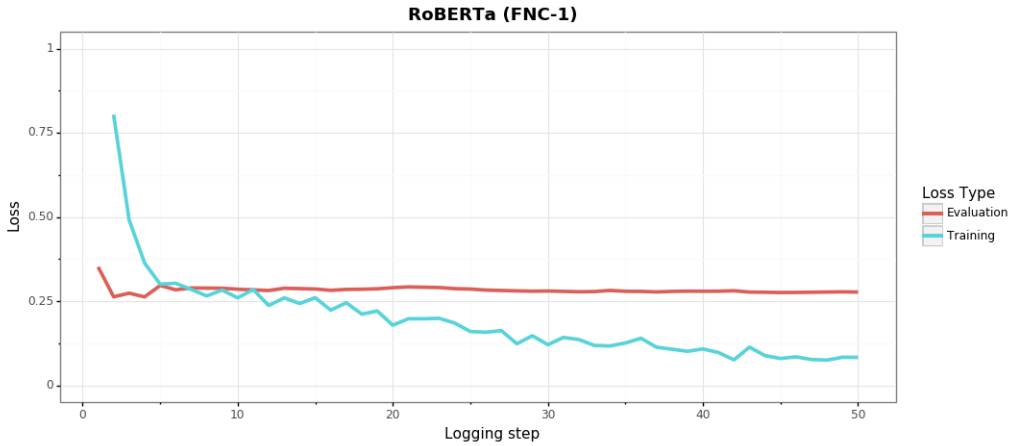

Fig. C.1: Overfitting plot for RoBERTa on the FNC-1 dataset. Every 50-th optimization step was tracked over the course of 2 epochs. There is no indication of overfitting as the evaluation loss stagnates.

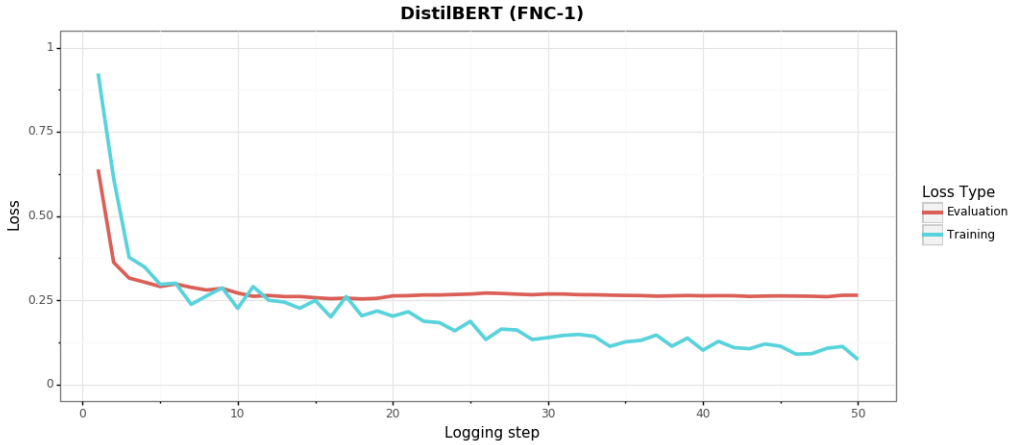

Fig. C.2: Overfitting plot for DistilBERT on the FNC-1 dataset. Every 50-th optimization step was tracked over the course of 2 epochs. There is no indication of overfitting as the evaluation loss stagnates.

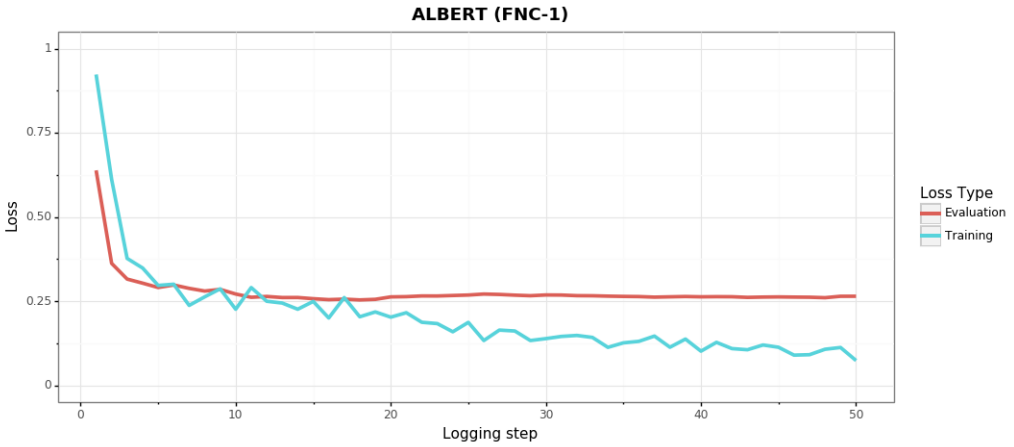

Fig. C.3: Overfitting plot for ALBERT on the FNC-1 dataset. Every 50-th optimization step was tracked over the course of 2 epochs. There is no indication of overfitting as the evaluation loss stagnates.

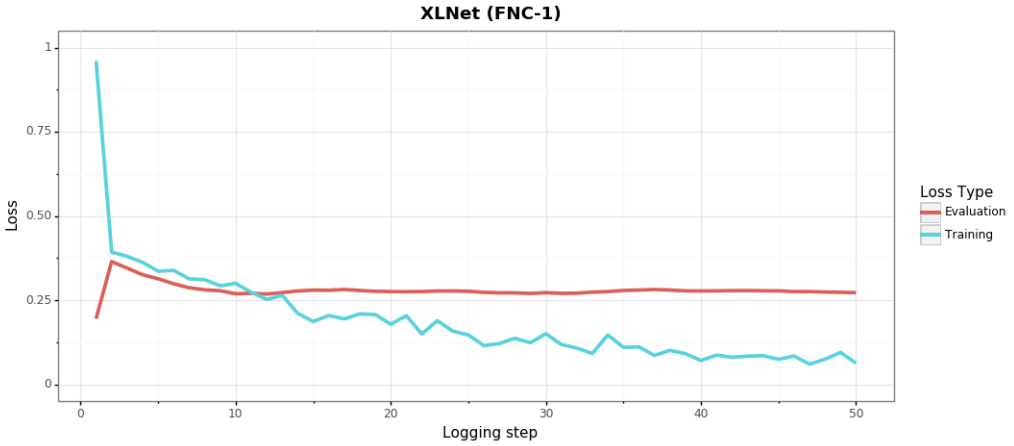

Fig. C.4: Overfitting plot for XLNet on the FNC-1 dataset. Every 50-th optimization step was tracked over the course of 2 epochs. There is no indication of overfitting as the evaluation loss stagnates.

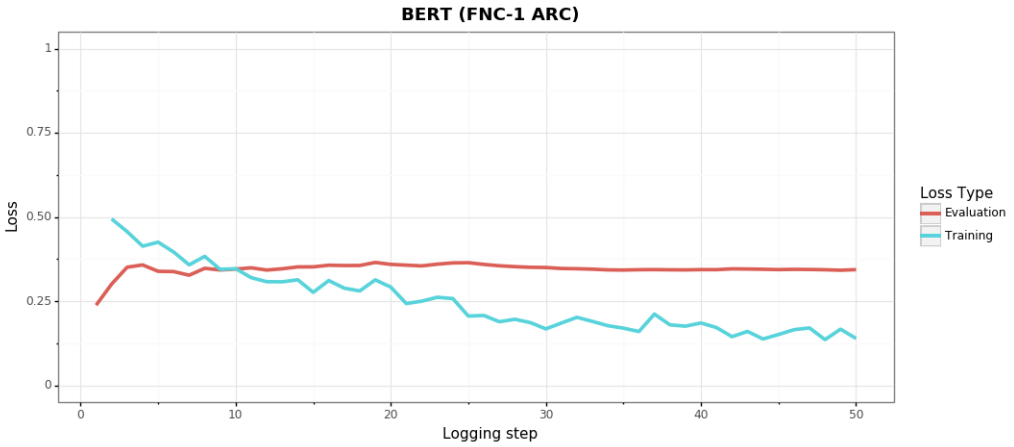

Fig. C.5: Overfitting plot for BERT on the FNC-1 ARC dataset. Every 50-th optimization step was tracked over the course of 2 epochs. There is no indication of overfitting as the evaluation loss stagnates.

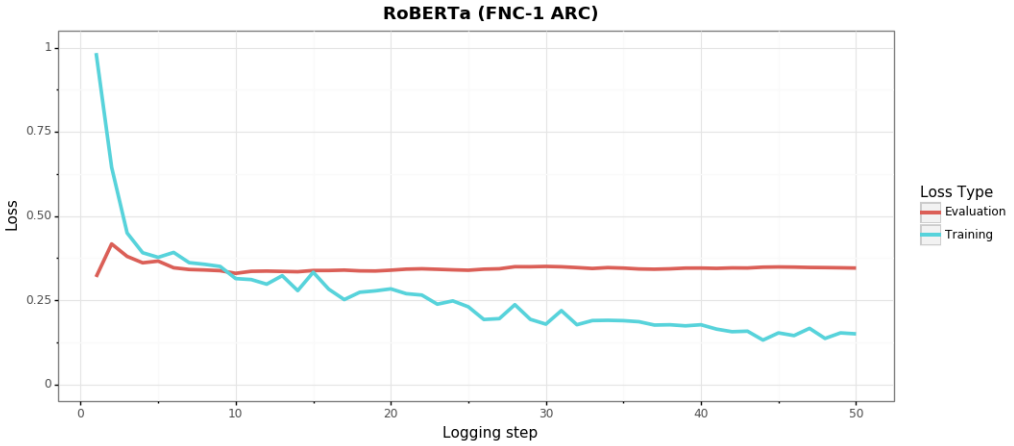

Fig. C.6: Overfitting plot for BERT on the FNC-1 ARC dataset. Every 50-th optimization step was tracked over the course of 2 epochs. There is no indication of overfitting as the evaluation loss stagnates.

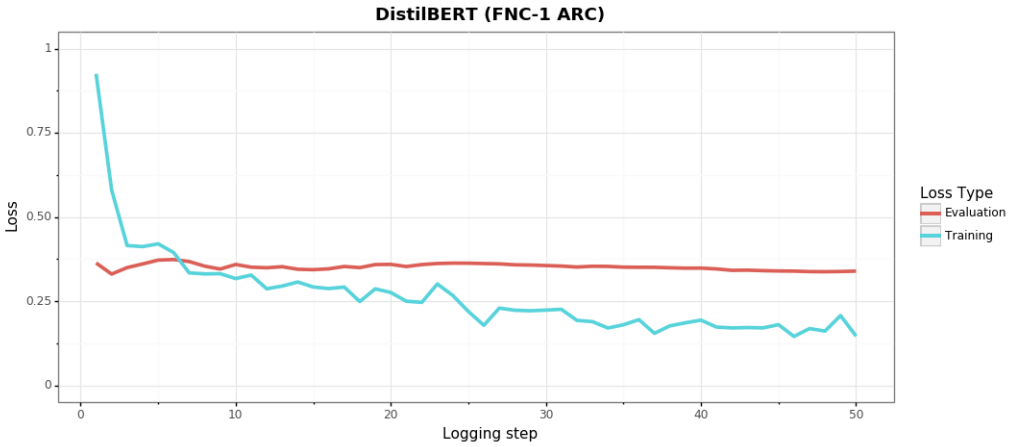

Fig. C.7: Overfitting plot for DistilBERT on the FNC-1 ARC dataset. Every 50-th optimization step was tracked over the course of 2 epochs. There is no indication of overfitting as the evaluation loss stagnates.

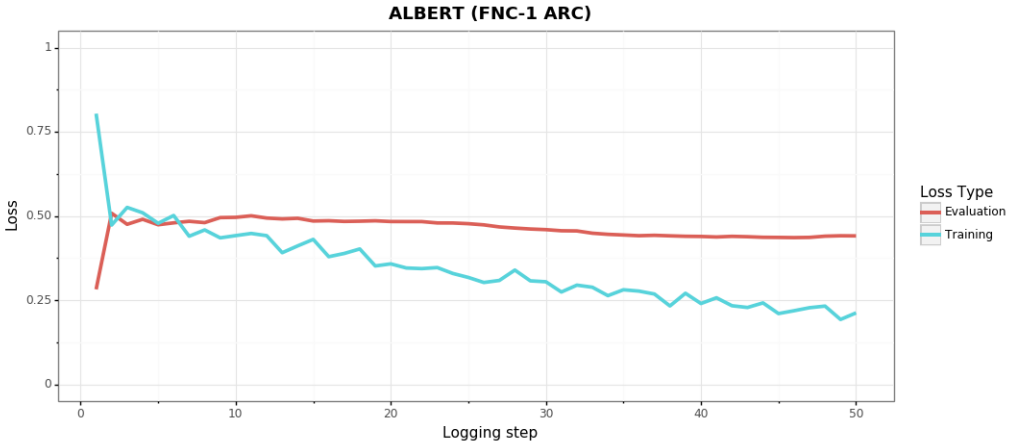

Fig. C.8: Overfitting plot for XLNet on the FNC-1 ARC dataset. Every 50-th optimization step was tracked over the course of 2 epochs. There is no indication of overfitting as the evaluation loss stagnates.

BERT FNC-1		True Class					
		Agree	Disagree	Discuss	Unrelated	$\sum$	F_1
Predicted Class	Agree	1,231	198	469	35	1,933	64.18
	Disagree	113	252	129	35	529	41.11
	Discuss	493	199	3,684	167	4,543	81.80
	Unrelated	66	48	182	18,112	18,408	98.55
	$\sum$	1,903	697	4,464	18,349	25,413	**71.41**

Table C.1: Confusion matrix of the final evaluation on the test set after 3 epochs for the winning configuration for BERT on the FNC-1 dataset. The winning configuration has a batch size of 16, sequence length of 256, learning rate of 2e-5 and a cosine learning rate schedule. The column "F_1" indicates the class-wise F_1 scores for rows 1 to 4 and the F_1-m value in bold which is 71.41.

RoBERTa FNC-1		True Class					
		Agree	Disagree	Discuss	Unrelated	$\sum$	F_1
Predicted Class	Agree	1,382	144	318	12	1,856	73.53
	Disagree	91	373	144	24	632	56.13
	Discuss	400	146	3,909	86	4,541	86.82
	Unrelated	30	34	93	18,227	18,384	99.24
	$\sum$	1,903	697	4,464	18,349	25,413	**78.93**

Table C.2: Confusion matrix of the final evaluation on the test set after 3 epochs for the winning configuration for RoBERTa on the FNC-1 dataset. The winning configuration has a batch size of 4, sequence length of 512, learning rate of 1e-5 and a linear learning rate schedule. The column "F_1" indicates the class-wise F_1 scores for rows 1 to 4 and the F_1-m value in bold which is 78.93. RoBERTa achieved the best overall result with respect to F_1-m on the FNC-1 dataset.

DistilBERT FNC-1		True Class					
		Agree	Disagree	Discuss	Unrelated	$\sum$	F_1
Predicted Class	Agree	1,256	240	342	24	1,862	66.72
	Disagree	68	174	117	4	363	32.83
	Discuss	523	223	3,833	125	4,704	83.62
	Unrelated	56	60	172	18,196	18,484	98.80
	$\sum$	1,903	697	4,464	18,349	25,413	**70.49**

Table C.3: Confusion matrix of the final evaluation on the test set after 3 epochs for the winning configuration for DistilBERT on the FNC-1 dataset. The winning configuration has a batch size of 32, sequence length of 256, learning rate of 4e-5 and a cosine learning rate schedule. The column "F_1" indicates the class-wise F_1 scores for rows 1 to 4 and the F_1-m value in bold which is 70.49.

ALBERT FNC-1		True Class					
		Agree	Disagree	Discuss	Unrelated	$\sum$	F_1
Predicted Class	Agree	1,049	209	597	75	1,930	54.74
	Disagree	53	130	63	7	253	27.37
	Discuss	514	203	3,244	173	4,134	75.46
	Unrelated	287	155	560	18,094	19,096	96.64
	$\sum$	1,903	697	4,464	18,349	25,413	**63.55**

Table C.4: Confusion matrix of the final evaluation on the test set after 3 epochs for the winning configuration for ALBERT on the FNC-1 dataset. The winning configuration has a batch size of 8, sequence length of 512, learning rate of 2e-5 and a cosine learning rate schedule. The column "F_1" indicates the class-wise F_1 scores for rows 1 to 4 and the F_1-m value in bold which is 63.55. ALBERT yielded the worst overall result with respect to F_1-m on the FNC-1 dataset.

XLNet FNC-1		True Class					
		Agree	Disagree	Discuss	Unrelated	$\sum$	F_1
Predicted Class	Agree	1,217	173	376	77	1,843	64.98
	Disagree	59	292	133	23	507	48.50
	Discuss	589	189	3,792	308	4,878	81.18
	Unrelated	38	43	163	17,941	18,185	98.22
	$\sum$	1,903	697	4,464	18,349	25,413	**73.22**

Table C.5: Confusion matrix of the final evaluation on the test set after 3 epochs for the winning configuration for XLNet on the FNC-1 dataset. The winning configuration has a batch size of 32, sequence length of 256, learning rate of 4e-5 and a constant learning rate schedule. The column "F_1" indicates the class-wise F_1 scores for rows 1 to 4 and the F_1-m value in bold which is 73.22.

BERT FNC-1 ARC		True Class					
		Agree	Disagree	Discuss	Unrelated	$\sum$	F_1
Predicted Class	Agree	1,386	278	505	27	2,196	62.53
	Disagree	199	434	194	32	859	45.02
	Discuss	553	276	3,685	245	4,759	78.39
	Unrelated	99	81	259	20,719	21,158	98.24
	$\sum$	2,237	1,069	4,643	21,023	28,972	**71.04**

Table C.6: Confusion matrix of the final evaluation on the test set after 3 epochs for the winning configuration for BERT on the FNC-1 ARC dataset. The winning configuration has a batch size of 16, sequence length of 256, learning rate of 2e-5 and a constant learning rate schedule. The column "F_1" indicates the class-wise F_1 scores for rows 1 to 4 and the F_1-m value in bold which is 71.04.

RoBERTa FNC-1 ARC		True Class					
		Agree	Disagree	Discuss	Unrelated	$\sum$	F_1
Predicted Class	Agree	1,564	229	407	21	2,221	70.17
	Disagree	141	512	126	10	789	55.11
	Discuss	503	289	3,995	178	4,965	83.16
	Unrelated	29	39	115	20,814	20,997	99.07
	$\sum$	2,237	1,069	4,643	21,023	28,972	**76.88**

Table C.7: Confusion matrix of the final evaluation on the test set after 3 epochs for the winning configuration for RoBERTa on the FNC-1 ARC dataset. The winning configuration has a batch size of 4, sequence length of 512, learning rate of 1e-5 and a cosine learning rate schedule. The column "F_1" indicates the class-wise F_1 scores for rows 1 to 4 and the F_1-m value in bold which is 76.88. RoBERTa achieved the best overall result with respect to F_1-m on the FNC-1 ARC dataset.

DistilBERT FNC-1 ARC		True Class					
		Agree	Disagree	Discuss	Unrelated	$\sum$	F_1
Predicted Class	Agree	1,489	347	439	33	2,308	65.52
	Disagree	132	422	124	26	704	47.60
	Discuss	543	227	3,930	190	4,890	82.45
	Unrelated	73	73	150	20,774	21,070	98.71
	$\sum$	2,237	1,069	4,643	21,023	28,972	**73.57**

Table C.8: Confusion matrix of the final evaluation on the test set after 3 epochs for the winning configuration for DistilBERT on the FNC-1 ARC dataset. The winning configuration has a batch size of 8, sequence length of 512, learning rate of 2e-5 and a linear learning rate schedule. The column "F_1" indicates the class-wise F_1 scores for rows 1 to 4 and the F_1-m value in bold which is 73.57.

ALBERT FNC-1 ARC		True Class					
		Agree	Disagree	Discuss	Unrelated	$\sum$	F_1
Predicted Class	Agree	1,412	393	885	104	2,794	56.13
	Disagree	87	251	82	21	441	33.25
	Discuss	386	189	3,155	270	4,000	73.01
	Unrelated	352	236	521	20,628	21,737	96.48
	$\sum$	2,237	1,069	4,643	21,023	28,972	**64.72**

Table C.9: Confusion matrix of the final evaluation on the test set after 3 epochs for the winning configuration for ALBERT on the FNC-1 ARC dataset. The winning configuration has a batch size of 4, sequence length of 512, learning rate of 1e-5 and a constant learning rate schedule. The column "F_1" indicates the class-wise F_1 scores for rows 1 to 4 and the F_1-m value in bold which is 64.72. ALBERT yielded the worst overall result with respect to F_1-m on the FNC-1 dataset.

XLNet FNC-1 ARC		True Class					
		Agree	Disagree	Discuss	Unrelated	$\sum$	F_1
Predicted Class	Agree	1,564	276	471	49	2,360	68.04
	Disagree	136	506	146	31	819	53.60
	Discuss	494	248	3,877	171	4,790	82.20
	Unrelated	43	39	149	20,772	21,003	98.85
	$\sum$	2,237	1,069	4,643	21,023	28,972	**75.67**

Table C.10: Confusion matrix of the final evaluation on the test set after 3 epochs for the winning configuration for XLNet on the FNC-1 ARC dataset. The winning configuration has a batch size of 4, sequence length of 512, learning rate of 1e-5 and a cosine learning rate schedule. The column "F_1" indicates the class-wise F_1 scores for rows 1 to 4 and the F_1-m value in bold which is 75.67.

Bibliography

Agarap, Abien Fred (2018). "Deep Learning using Rectified Linear Units (ReLU)". In: *CoRR* abs/1803.08375.

Ba, Jimmy, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). "Layer Normalization". In: *ArXiv* abs/1607.06450.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *ArXiv* 1409.0473.

Bengio, Yoshua et al. (2003). "A Neural Probabilistic Language Model". In: *Journal of Machine Learning Research 3* abs/1706.03762, pp. 1137–1155.

Benjamin Riedel et al. (2017a). "A simple but tough-to-beat baseline for the Fake News Challenge stance detection task". In: *CoRR* abs/1707.03264.

– (2017b). *Webpage of the third winning team of the FNC-1*. `https://web.archive.org/web/20200220104224/https://github.com/uclnlp/fakenewschallenge`. Accessed: 2020-02-20.

BuzzFeed (2020). *BuzzFeedNews Dataset*. `https://github.com/BuzzFeedNews/2016-10-facebook-fact-check/tree/master/data`. Accessed: 2020-03-03.

Channel 4 (2020). *Channel 4 News: Fact Check*. `https://www.channel4.com/news/factcheck/`. Accessed: 2020-03-01.

Chen, Tianqi and Carlos Guestrin (2016). "XGBoost: A Scalable Tree Boosting System". In: *CoRR* abs/1603.02754.

Clark, Kevin et al. (2019). "What Does BERT Look At? An Analysis of BERT's Attention". In: *CoRR* abs/1906.04341.

Dai, Zihang et al. (2019). "Transformer-XL: Attentive Language Models beyond a Fixed-Length Context". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics* 10.18653/v1/p19-1285.

Devlin, Jacob et al. (2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805.

Dong, Xishuang et al. (2019). "Deep Two-path Semi-supervised Learning for Fake News Detection". In: *ArXiv* 1906.05659.

Ferreira, William and Andreas Vlachos (2016). "Emergent: a novel data set for stance detection". In: *Proceedings of NAACL-HLT 201*, pp. 1163–1168.

Goldberg, Yoav and Graeme Hirst (2017). *Neural Network Methods in Natural Language Processing*. Morgan Claypool Publishers.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press.

Hanselowski, Andreas et al. (2018). "A Retrospective Analysis of the Fake News Challenge Stance Detection Task". In: *CoRR* abs/1806.05180.

Hao, Karen (2019). *Training a single AI model can emit as much carbon as five cars in their lifetimes*. https://www.technologyreview.com/s/613630/training-a-single-ai-model-can-emit-as-much-carbon-as-five-cars-in-their-lifetimes/. Accessed: 2020-02-18.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *"The Elements of Statistical Learning: Data Mining, Inference and Prediction"*. 2nd ed. Springer.

He, Kaiming et al. (2015). "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385.

Hendrycks, Dan and Kevin Gimpel (2016). "Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units". In: *CoRR* abs/1606.08415.

Hinton, Geoffrey E., Oriol Vinyals, and Jeffrey Dean (2015). "Distilling the Knowledge in a Neural Network". In: *ArXiv* abs/1503.02531.

HuggingFace (2020a). *HuggingFace Training Script Example)*. https://web.archive.org/save/https://github.com/huggingface/transformers/tree/master/examples. Accessed: 2020-02-20.

– (2020b). *Webpage of HuggingFace with Learning Rate Schedules*. https://web.archive.org/save/https://huggingface.co/transformers/main_classes/optimizer_schedules.html#schedules. Accessed: 2020-24-02.

Joshi, Mandar et al. (2019). "SpanBERT: Improving Pre-training by Representing and Predicting Spans". In: *CoRR* abs/1907.10529.

Jurafsky, Daniel and James H. Martin (2019). *Speech and Language Processing: An Introduction to Natural Language Processing*. 3rd. Computational Linguistics and Speech Recognition. Prentice Hall.

Kahneman, Daniel and Amos Tversky (1979). "Prospect Theory: An Analysis of Decision under Risk". In: *Econometrica* 47.2, pp. 263–291.

Khan, Junaed Younus et al. (2019). "A Benchmark Study on Machine Learning Methods for Fake News Detection". In: *CoRR* abs/1905.04749.

Kingma, Diederik P. and Jimmy Ba (2014). "Adam: A Method for Stochastic Optimization". In: *ArXiv* 1412.6980.

Kudo, Taku and John Richardson (2018). "SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*.

Lai, Guokun et al. (2017). "RACE: Large-scale ReAding Comprehension Dataset From Examinations". In: *ArXiv* 1704.04683.

Lan, Zhenzhong et al. (2019). "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations". In: *ArXiv* 1909.11942.

Lemur Project (2020). *ClueWeb09 Dataset*. https://lemurproject.org/clueweb09.php/. Accessed: 2020-03-03.

Li, Hao et al. (2020). “Rethinking the Hyperparameters for Fine-tuning”. In: *International Conference on Learning Representations*.

Liaw, Richard et al. (2018). “Tune: A Research Platform for Distributed Model Selection and Training”. In: *arXiv* 1807.05118.

Liu, Dianbo and Tim Miller (2020). “Federated pretraining and fine tuning of BERT using clinical notes from multiple silos”. In: *ArXiv* 2002.08562.

Liu, Xinyi and Artit Wangperawong (2019). “Transfer Learning Robustness in Multi-Class Categorization by Fine-Tuning Pre-Trained Contextualized Language Models”. In: *ArXiv* 1909.03564.

Liu, Yang (2019). “Fine-tune BERT for Extractive Summarization”. In: *ArXiv* 1903.10318.

Liu, Yinhan et al. (2019). “RoBERTa: A Robustly Optimized BERT Pretraining Approach”. In: *CoRR* abs/1907.11692.

Loshchilov, Ilya and Frank Hutter (2018). “Fixing Weight Decay Regularization in Adam”. In: URL: https://openreview.net/forum?id=rk6qdGgCZ.

lrz (2020). *lrz Compute Cloud*. https://web.archive.org/web/20200220142228/https://doku.lrz.de/display/PUBLIC/Compute+Cloud. Accessed: 2020-01-09.

Michel, Paul, Omer Levy, and Graham Neubig (2019). “Are Sixteen Heads Really Better than One?” In: *CoRR* abs/1905.10650.

Mikolov, Tomas et al. (2013). “Distributed Representations of Words and Phrases and their Compositionality”. In: *CoRR* abs/1310.4546.

Mitra, Tanushree and Eric Gilbert (2015). “CREDBANK: A Large-Scale Social Media Corpus With Associated Credibility Annotations”. In: *International AAAI Conference on Web and Social Media* ICWSM15.

Mueller, Special Counsel Robert S. (2019). *Report On The Investigation Into Russian Interference In The 2016 Presidential Election*. Tech. rep. U.S. Department of Justice.

Ni, Bo et al. (2020). “Improving Generalizability of Fake News Detection Methods using Propensity Score Matching”. In: *ArXiv* 2002.00838.

Nothman, Joel, Hanmin Qin, and Roman Yurchak (2018). “Stop Word Lists in Free Open-source Software Packages”. In: *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*. Melbourne, Australia: Association for Computational Linguistics, pp. 7–12.

Nyhan, Brendan and Jason Reifler (2010). “When Corrections Fail: The Persistence of Political Misperceptions”. In: *Political Behavior* 32.2, pp. 303–330.

Pan, Yuxi, Doug Sibley, and Sean Baird (2017). *Webpage of the winning team of the FNC-1*. https://web.archive.org/web/20200220101512/https://github.com/Cisco-Talos/fnc-1. Accessed: 2020-02-20.

Paszke, Adam et al. (2019). “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *ArXiv* 1912.01703.

Pedregosa, F. et al. (2011). “Scikit-learn: Machine Learning in Python ”. In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Peng, Tony (2019). *The Staggering Cost of Training SOTA AI Models*. https://web.archive.org/save/https://syncedreview.com/2019/06/27/the-staggering-cost-of-training-sota-ai-models/. Accessed: 2020-02-20.

Peng, Xiangyu et al. (2020). "Fine-Tuning a Transformer-Based Language Model to Avoid Generating Non-Normative Text". In: *ArXiv* 2001.08764.

Pennington, Jeffrey, Richard Socher, and Christopher Manning (2014). "Glove: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1532–1543.

Pérez-Rosas, Verónica et al. (2018). "Automatic Detection of Fake News". In: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 3391–3401.

Peters, Matthew et al. (2018). "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*.

Petroni, Fabio et al. (2019). "Language Models as Knowledge Bases?" In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Pomerleau, Dean and Delip Rao (2017). *Webpage of the Fake News Challenge 1 (FNC-1)*. https://web.archive.org/save/http://www.fakenewschallenge.org/. Accessed: 2020-02-20.

Powell, Rob (2019). *Google BERT algorithm (& why there's nothing you can do about it)*. https://web.archive.org/save/https://robpowellbizblog.com/google-bert-algorithm/. Accessed: 2020-02-20.

Prof. Dr. Gurevych, Iryna (2017). *Webpage of the second winning team of the FNC-1*. https://web.archive.org/web/20200220102827/https://github.com/hanselowski/athene_system. Accessed: 2020-02-20.

Radford, Alec et al. (2019). *Language Models are Unsupervised Multitask Learners*.

Rajpurkar, Pranav et al. (2016). "SQuAD: 100, 000+ Questions for Machine Comprehension of Text". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

Reitan, Johan et al. (2015). "Negation Scope Detection for Twitter Sentiment Analysis". In: *Proceedings of the 6th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*. Lisboa, Portugal: Association for Computational Linguistics, pp. 99–108.

Ren, Yuxiang and Jiawei Zhang (2020). "HGAT: Hierarchical Graph Attention Network for Fake News Detection". In: *ArXiv* 2002.04397.

Rosenbaum, Paul R. and Donald B. Rubin (1983). "The central role of the propensity score in observational studies for causal effects". In: *Biometrika* 70.1, pp. 41–55.

Rossum, Guido van (1995). *Python Reference Manual*. Tech. rep. CS-R9526. Amsterdam: Centrum voor Wiskunde en Informatica (CWI).

Ruchansky, Natali, Sungyong Seo, and Yan Liu (2017). “CSI: A Hybrid Deep Model for Fake News”. In: *CoRR* abs/1703.06959.

Sanh, Victor et al. (2019). “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. In: *ArXiv* 1910.01108.

Santia, Giovanni and Jake Williams (2018). “BuzzFace: A News Veracity Dataset with Facebook User Commentary and Egos”. In: URL: https://aaai.org/ocs/index.php/ICWSM/ICWSM18/paper/view/17825.

Shu, Kai (2020). *FakeNewsNet GitHub Repository*. https://github.com/KaiDMML/FakeNewsNet. Accessed: 2020-03-01.

Shu, Kai, Deepak Mahudeswaran, et al. (2018). “FakeNewsNet: A Data Repository with News Content, Social Context and Spatialtemporal Information for Studying Fake News on Social Media”. In: *ArXiv* 1809.01286.

Shu, Kai, Amy Sliva, et al. (2017). “Fake news detection on social media: A data mining perspective”. In: *ACM SIGKDD Explorations Newsletter* 19.1, pp. 22–36.

Shuster, Andrew (2020). *Gossip Cop*. https://www.gossipcop.com/. Accessed: 2020-03-01.

Sieradski, Daniel (2020). *B.S. Detector Dataset*. https://gitlab.com/bs-detector/bs-detector. Accessed: 2020-03-03.

Silverman, Craig (2019). *Emergent: A real-time tracker*. http://www.emergent.info/. Accessed: 2019-12-17.

Srivastava, Nitish et al. (2014). “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15, pp. 1929–1958.

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le (2014). *Sequence to Sequence Learning with Neural Networks*.

Tacchini, Eugenio et al. (2017). *Some Like it Hoax: Automated Fake News Detection in Social Networks*.

Tang, Raphael et al. (2019). “Distilling Task-Specific Knowledge from BERT into Simple Neural Networks”. In: *CoRR* abs/1903.12136.

Tenney, Ian, Dipanjan Das, and Ellie Pavlick (2019). “BERT Rediscovers the Classical NLP Pipeline”. In: *CoRR* abs/1905.05950.

Thanaki, Jalaj (2017). *Python Natural Language Processing*. Packt Publishing Ltd.

Thorne, James et al. (2018). “FEVER: a large-scale dataset for Fact Extraction and VERification”. In: *CoRR* abs/1803.05355.

Times, Tampa Bay (2020). *Politifact: Truth-O-Meter*. https://www.politifact.com/truth-o-meter/. Accessed: 2020-03-01.

Vasandani, Jasmine (2019). *Building a Fake News Detector*. https://towardsdatascience.com/i-built-a-fake-news-detector-using-natural-language-processing-and-classification-models-da180338860e. Accessed: 2020-02-29.

Vaswani, Ashish et al. (2017). “Attention Is All You Need”. In: *CoRR* abs/1706.03762.

Vlachos, Andreas and Sebastian Riedel (2014). "Fact Checking: Task definition and dataset construction". In: *Proceedings of the ACL 2014 Workshop on Language Technologies and Computational Social Science*. Baltimore, MD, USA: Association for Computational Linguistics.

Wang, Alex et al. (2018). "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*.

Wang, William Yang (2017). ""Liar, Liar Pants on Fire": A New Benchmark Dataset for Fake News Detection". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*.

Wang, Yu et al. (2020). "Application of Pre-training Models in Named Entity Recognition". In: *ArXiv* 2002.08902.

Weedon, Jen, William Nuland, and Alex Stamos (2017). *Information Operations and Facebook*. `https://fbnewsroomus.files.wordpress.com/2017/04/facebook-and-information-operations-v1.pdf`. Accessed: 2020-02-29.

Wolf, Thomas et al. (2019). "HuggingFace's Transformers: State-of-the-art Natural Language Processing". In: *ArXiv* 1910.03771.

Wu, Yonghui et al. (2016). "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *CoRR* abs/1609.08144.

Yang, Yang et al. (2018). "TI-CNN: Convolutional Neural Networks for Fake News Detection". In: *ArXiv* 1806.00749.

Yang, Zhilin et al. (2019). "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *CoRR* abs/1906.08237.

Yoon, Seunghyun et al. (2018). "Detecting Incongruity Between News Headline and Body Text via a Deep Hierarchical Encoder". In: *CoRR* abs/1811.07066.

You, Yang, Igor Gitman, and Boris Ginsburg (2017). "Large Batch Training of Convolutional Networks". In: *ArXiv* 1708.03888.

You, Yang, Jing Li, et al. (2019). "Large Batch Optimization for Deep Learning: Training BERT in 76 Minutes". In: *CoRR* abs/1904.00962.

Zellers, Rowan, Yonatan Bisk, et al. (2018). "SWAG: A Large-Scale Adversarial Dataset for Grounded Commonsense Inference". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Zellers, Rowan, Ari Holtzman, et al. (2019). "Defending Against Neural Fake News". In: *CoRR* abs/1905.12616.